SOURCE DYNAMICS
SOURCE INSIGHT VERSION 3.*x* END-USER LICENSE AGREEMENT

IMPORTANT-READ CAREFULLY: This Source Insight End-User License Agreement ("EULA") is a legal agreement between you (either an individual or a single entity) and Source Dynamics, Inc. for the Source Dynamics SOFTWARE identified above, which includes the User Manual, any associated SOFTWARE components, any media, any printed materials other than the User Manual, and any "online" or electronic documentation ("SOFTWARE"). By installing, copying, or otherwise using the SOFTWARE, you agree to be bound by the terms of this EULA. If you do not agree to the terms of this EULA, do not install or use the SOFTWARE. If the SOFTWARE was mailed to you, return the media envelope, UNOPENED, along with the rest of the package to the location where you obtained it within 30 days from purchase.

1. The SOFTWARE is licensed, not sold.

2. GRANT OF LICENSE.

(a) Evaluation Copy. You may use the SOFTWARE without charge on an evaluation basis for thirty (30) days from the day that you install the SOFTWARE. You must pay the license fee and register your copy to continue to use the SOFTWARE after the thirty (30) days. If you continue to use the SOFTWARE after the thirty (30) days without paying the license fee you will be using the SOFTWARE on an unlicensed basis.

(b) Redistribution of Evaluation Copy. If you are using SOFTWARE on an evaluation basis you may make copies of the evaluation SOFTWARE as you wish; give exact copies of the original evaluation SOFTWARE to anyone; and distribute the evaluation SOFTWARE in its unmodified form via electronic means (Internet, BBS's, Shareware distribution libraries, CD-ROMs, etc.). You may not charge any fee for the copy or use of the evaluation SOFTWARE itself, but you may charge a distribution fee that is reasonably related to any cost you incur distributing the evaluation SOFTWARE (e.g. packaging). You must not represent in any way that you are selling the SOFTWARE itself. Your distribution of the evaluation SOFTWARE will not entitle you to any compensation from Source Dynamics. You must distribute a copy of this EULA with any copy of the SOFTWARE and anyone to whom you distribute the SOFTWARE is subject to this EULA.

(c) Registered Copy. After you have purchased the license for SOFTWARE, and have received the serial number enabling the registered copy, you are licensed to copy the SOFTWARE only into the memory of the number of computers corresponding to the number of licenses purchased. The primary user of the computer on which each licensed copy of the SOFTWARE is installed may make a second copy for his or her exclusive use on a portable computer. Under no other circumstances may the SOFTWARE be operated at the same time on more than the number of computers for which you have paid a separate license fee. You may not duplicate the SOFTWARE in whole or in part, except that you may make one copy of the SOFTWARE for backup or archival purposes. You may terminate this license at any time by destroying the original and all copies of the SOFTWARE in whatever form. You may permanently transfer all of your rights under this EULA provided you transfer all copies of the SOFTWARE (including copies of all prior versions if the SOFTWARE is an upgrade) and retain none, and the recipient agrees to the terms of this EULA.

3. RESTRICTIONS. You may not reverse engineer, de-compile, or disassemble the SOFTWARE, except and only to the extent that such activity is expressly permitted by applicable law notwithstanding this limitation. You may not rent, lease, or lend the SOFTWARE. You may permanently transfer all of your rights under this EULA, provided the recipient agrees to the terms of this EULA. You may not publish or publicly distribute any serial numbers, access codes, unlock-codes, passwords, or other end-user-specific registration information that would allow a third party to activate the SOFTWARE without a valid license.

4. SUPPORT SERVICES. Source Dynamics may provide you with support services related to the SOFTWARE. Use of Support Services is governed by the Source Dynamics polices and programs described in the user manual, in online documentation, and/or other Source Dynamics-provided materials, as they may be modified from time to time. Any supplemental SOFTWARE code provided to you as part of the Support Services shall be considered part of the SOFTWARE and subject to the terms and conditions of this EULA.

5. TERMINATION. Without prejudice to any other rights, Source Dynamics may terminate this EULA if you fail to comply with the terms and conditions of this EULA. In such event, you must destroy all copies of the SOFTWARE.

6. COPYRIGHT. The SOFTWARE is protected by United States copyright law and international treaty provisions. You acknowledge that no title to the intellectual property in the SOFTWARE is transferred to you. You further acknowledge that title and full ownership rights to the SOFTWARE will remain the exclusive property of Source Dynamics and you will not acquire any rights to the SOFTWARE except as expressly set forth in this license. You agree that any copies of the SOFTWARE will contain the same proprietary notices which appear on and in the SOFTWARE.

7. EXPORT RESTRICTIONS. You agree that you will not export or re-export the SOFTWARE to any country, person, entity, or end user subject to U.S.A. export restrictions. Restricted countries currently include, but are not necessarily limited to Cuba, Iran, Iraq, Libya, North Korea, Sudan, and Syria. You warrant and represent that neither the U.S.A. Bureau of Export Administration nor any other federal agency has suspended, revoked or denied your export privileges.

8. LIMITED WARRANTY. Source Dynamics, Inc. warrants that the Software will perform substantially in accordance with the accompanying written materials for a period of 90 days from the date of your receipt of the Software. Any implied warranties on the Software are limited to 90 days. Some states do not allow limitations on duration of an implied warranty, so the above limitation may not apply to you. SOURCE DYNAMICS, INC. DISCLAIMS ALL OTHER WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT, WITH RESPECT TO THE SOFTWARE AND THE ACCOMPANYING WRITTEN MATERIALS. This limited warranty gives you specific legal rights. You may have others, which vary from state to state.

9. LIMITATION OF LIABILITY. IN NO EVENT SHALL SOURCE DYNAMICS OR ITS SUPPLIERS BE LIABLE TO YOU FOR ANY CONSEQUENTIAL, SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES OF ANY KIND ARISING OUT OF THE DELIVERY, PERFORMANCE, OR USE OF THE SOFTWARE, EVEN IF SOURCE DYNAMICS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN ANY EVENT, SOURCE DYNAMICS'S LIABILITY FOR ANY CLAIM, WHETHER IN CONTRACT, TORT, OR ANY OTHER THEORY OF LIABILITY WILL NOT EXCEED THE GREATER OF U.S.$1.00 OR LICENSE FEE PAID BY YOU.

10. U.S. GOVERNMENT RESTRICTED RIGHTS. The SOFTWARE is provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph ©(1)(ii) of The Rights in Technical Data and Computer SOFTWARE clause of DFARS 252.227-7013 or subparagraphs ©(i) and (2) of the Commercial Computer SOFTWARE-Restricted Rights at 48 CFR 52.227-19, as applicable. Manufacturer is Source Dynamics, Inc., 22525 SE 64th Place, Suite 260, Issaquah, WA 98027, USA.

11. MISCELLANEOUS. If you acquired the SOFTWARE in the United States, this EULA is governed by the laws of the state of Washington. If you acquired the SOFTWARE outside of the United States, then local laws may apply.

If you have any questions concerning this EULA or wish to contact Source Dynamics for any reason, please write: Source Dynamics, Inc., 22524 SE 64th Place, Suite 260, Issaquah, WA 98027, USA; or call (425) 557-3630; or send electronic mail to support@sourceinsight.com.

Source Insight 3.1 User Manual

Copyright © 2001 by Source Dynamics, Inc.

8.21.2001

Source Insight - Version 3.1
Copyright © 1987-2001 by Source Dynamics, Inc.

This documentation and the software described in it are copyrighted with all rights reserved. Under the copyright laws, neither the documentation nor the software may be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form, in whole or in part, without the prior written consent of Source Dynamics, Inc.

Disclaimer of Warranties and Limitation of Liabilities

This manual and the software described in it were prepared by Source Dynamics, Inc., and are subject to change without notice. Source Dynamics, Inc. assumes no liability resulting from any inaccuracy or omissions contained herein or from the use of the information or programs contained herewith.

Source Dynamics, Inc. makes no expressed or implied warranty of any kind with regard to these programs or to the supplemental documentation in this manual. In no event shall Source Dynamics, Inc. be liable for incidental or consequential damages in connection with or arising out of the furnishing, performance, or use of the program or documentation. This disclaimer includes, but is not limited to, any loss of service, loss of business, or anticipatory profits, or consequential damages resulting from the use or operation of the enclosed software.

Source Insight is a trademark of Source Dynamics, Inc.
Other product names are trademarks of their respective manufacturers.

Source Dynamics, Inc.
22525 SE 64th Place, Suite 260
Issaquah, WA 98027
USA

Phone: (425) 557-3630
Fax: (425) 557-3631
Web: www.sourceinsight.com

Technical Support: support@sourceinsight.com

Sales: sales@sourceinsight.com

Contents

Introduction 23

The Big Picture 24
- The Problem 24
- The Solution 24

Feature Highlights 26

What's New in Version 3 39
- Improved Language Features 39
- Improved Browsing and Analysis Features 40
- Improved Editing and Display Features 41

Setup and Quick Start 43

Installing Source Insight 44
- Installing on Windows NT/2000/XP 44
- Insert the CD-ROM 44
- Choosing a Drive for the Installation 44
- Upgrading from Version 2 45
- Configuring Source Insight 45
- Entering Your Serial Number 46
- Creating Common Projects 46
- Creating a Project 46
- Where to Go Next? 47

Window Tour 49

Main Source Insight Window 50
- Floating Windows 50
- Toolbars 51

Source File Window 56

Symbol Window 57
- Selecting Symbols Quickly 58
- Typing into the Symbol Window 58
- Rearranging Code in the Symbol Window 58
- Customizing the Symbol Window 59

Project Window 60
- Project Window Views 61

Context Window 65

Previewing Files 65
Showing Declarations 66
Decoding Base Types to Show Structures 67
Customizing the Context Window 68
Relation Window 69
Outline and Graph Views 69
Relationship Types 69
Call Graphs 70
Multiple Relation Windows 71
Relation Window Properties 71
Clip Window 72
What Is A Clip? 72
Creating a New Clip 72
Clip Storage 73
Search Results Window 74

Source Insight Concepts 77

Projects 78
The Current Project 79
How Source Insight Uses Projects 79
Creating a Project 80
Normalized File Names 81
The Project List 81
Adding Files to a Project 82
Removing Files from a Project 82
Closing Projects 83
Opening Projects 83
Removing a Project 83
Project Settings 83
Working in a Team 85
Networking 85
Source Control 86
Symbols and Projects 88
File Names Are Like Symbols 88
Synchronizing Project Files 89
The Project Symbol Path 89
The Base Project 91
Language Support 92
Built-In Languages 92
Custom Languages 92
HTML Parsing 93
HTML and ASP Compound Languages 93
C/C++ Issues 94
Preprocessor Token Macros 96
Parsing Considerations 97
Document Types 100
Document-Specific Options 100

Associating Files with Document Types 101
Adding Known File Types 101
Editing the Document Options 102
Symbol Syllable Indexing 103
What is a Symbol Syllable? 103
Symbol Indexes for Projects 104
Index Options for Projects 104
Controlling Syllable Matching 105
Using Syllable Matching 105
Accessing Symbol Information 107
Background Parsing 107
Jump to Definition command 107
Jump to Caller command 107
Browse Project Symbols command 108
Browse Local File Symbols command 108
Project Window Symbol List 108
Call Trees and Reference Trees 108
Context Window 109
Command Line 109
Finding References to Symbols 109
Project Report 109
Smart Renaming 110
Syntax Formatting and Styles 111
How a Style Works 111
How Styles Apply to Source Code 113
Syntax Decorations 116
Controlling Syntax Formatting 117
Selection Shortcuts 120
Selecting Whole Words 120
Selecting Whole Functions or Symbols 120
Selecting Matching Parentheses and Blocks 120
Selecting the Enclosing Block 120
Selecting a Whole Line 121
Selecting a Paragraph of Text 121
Selecting Between Lines 121
Searching and Replacing Text 122
Searching for Symbol References 122
Renaming an Identifier 122
Searching the Current File 123
Replacing in the Current File 123
Searching Multiple Files 123
Replacing in Multiple Files 123
Searching for Keywords 124
Regular Expressions 125
Wildcard Matching 125
Matching the Beginning or End of a Line 125
Matching a Tab or Space 125
Matching 0, 1, or More Occurrences 126

Matching Any in a Set of Characters 126
Regular Expression Groups 127
Overriding Regular Expression Characters 127
Regular Expression Summary 128
Bookmarks 129
Selection History 130
Go Back and Go Forward commands 130
Source Links 132
Searching and Source Links 132
Creating Source Links 133
Source Links from Custom Command Output 133
Navigating with Source Links 134
Scrolling and Selecting Text 135
Moving Through a File 135
Using the Scrolling Commands 135
Using the Selection Commands 136
Extending the selection 138
How Source Insight Edits Files 141
Time stamping 142
What Happens when you Start Source Insight 142
Recovering From Crashes 143
Recovery Procedure 143
The Command Line 145
File Arguments 145
Opening Files 145
File search path 146
Opening Workspaces 146
Command Line Options 147
Commands in Source Insight 150
Custom Commands 151
Customizing Source Insight 152
Configurations 154
Configuration Files 154
Loading a Configuration 156
Saving a Configuration 156
Workspaces 157
Loading and Saving Workspaces 157
Managing Tasks With Workspaces 158
Performance Tuning 159
Factors That Affect Performance 159
Speeding Up Program Features 162
Files Created by Source Insight 167
Files in the Program Directory 167
Files Created for Each Project 168
User Data Directories 168

Command Reference 171

Commands 172
About Source Insight 173
Activate Menu Commands 173
Activate Global Symbol List 173
Activate Relation Window 174
Activate Search Results 174
Activate Symbol Window 174
Add and Remove Project Files 174
Add File 177
Add File List 179
Advanced Options 179
Back Tab 179
Backspace 179
Beginning of Line 179
Beginning of Selection 180
Blank Line Down 180
Blank Line Up 180
Block Down 180
Block Up 180
Bookmark 180
Bottom of File 181
Bottom of Window 182
Browse Files 182
Browse Project Symbols 182
Browse Local File Symbols 187
Cascade Windows 190
Checkpoint 190
Checkpoint All 190
Clear Highlights 190
Clip Properties 190
Clip Window Properties 190
Close 192
Close All 192
Close Project 192
Close Window 193
Color Options 193
Command Shell 194
Complete Symbol 194
Context Window 195
Context Window Properties 195
Copy 197
Copy Line 197
Copy Line Right 198
Copy List 198
Copy Symbol 198
Copy To Clip 198
Create Key List 199
Create Command List 199

Cursor Down 199
Cursor Left 199
Cursor Right 200
Cursor Up 200
Custom Commands 200
Cut 209
Cut Line 209
Cut Line Left 209
Cut Line Right 209
Cut Selection or Paste 209
Cut Symbol 210
Cut To Clip 210
Cut Word 210
Cut Word Left 210
Delete 210
Delete All Clips 210
Delete Character 211
Delete Clip 211
Delete File 211
Display Options 211
Document Options 219
Draft View 226
Drag Line Down 227
Drag Line Down More 227
Drag Line Up 227
Drag Line Up More 227
Duplicate 227
Duplicate Symbol 227
Edit Condition 227
End of Line 229
End of Selection 229
Exit 229
Exit and Suspend 230
Expand Special 230
File Options 231
Function Down 234
Function Up 234
General Options 234
Go Back 237
Go Back Toggle 238
Go Forward 238
Go To First Link 238
Go To Line 240
Go To Next Change 240
Go To Previous Change 240
Go To Next Link 241
Go To Previous Link 241

Help 241
Help Mode 241
Highlight Word 241
Incremental Search 242
Incremental Search Backward 243
Horizontal Scroll Bar 243
HTML Help 243
Indent Left 243
Indent Right 243
Insert ASCII 243
Insert File 244
Insert Line 246
Insert Line Before Next 246
Insert New Line 246
Join Lines 246
Jump To Base Type 247
Jump To Caller 247
Jump To Definition 247
Jump To Link 248
Jump To Prototype 248
Key Assignments 248
Keyword List 251
Language Options 254
Language Properties 258
Line Numbers 265
Link All Windows 266
Link Window 266
Load Configuration 267
Load File 269
Load Search String 272
Lock Context Window 272
Lock Relation Window 272
Lookup References 272
Make Column Selection 277
Menu Assignments 277
New 279
New Clip 279
New Relation Window 279
New Project 280
New Window 281
Next File 281
Next Relation Window View 281
Open 281
Open Project 282
Page Down 283
Page Setup 284
Page Up 286
Paren Left 286

Paren Right 287
Parse Source Links 287
Paste 288
Paste From Clip 288
Paste Line 288
Paste Symbol 288
Play Recording 289
Preferences 289
Project Document Types 290
Project File Browser 290
Project File List 291
Project Symbol Classes 291
Project Symbol List 291
Project Window Properties 292
Print 293
Print Relation Window 294
Project Settings 294
Project Report 298
Project Window command 299
Rebuild Project 299
Record New Default Properties 301
Redo 301
Redo All 301
Redraw Screen 302
Reform Paragraph 302
Refresh Relation Window 302
Relation Graph Properties 303
Relation Window 304
Relation Window Properties 305
Reload File 309
Reload Modified Files 309
Remove File 310
Remove Project 311
Rename 313
Renumber 313
Repeat Typing 314
Replace 314
Replace Files 317
Restore File 320
Restore Lines 320
Save 321
Save A Copy 321
Save All 321
Save All Quietly 322
Save As 322
Save Configuration 323
Save Selection 324

Save Workspace 324
Scroll Half Page Down 325
Scroll Half Page Up 325
Scroll Left 325
Scroll Line Down 325
Scroll Line Up 325
Scroll Right 325
SDK Help 325
Search 326
Search Backward 328
Search Backward for Selection 328
Search Files 328
Search Forward 332
Search Forward for Selection 332
Search List 333
Search Project 333
Searching Options 333
Select All 334
Select Block 334
Select Char Left 335
Select Char Right 335
Select Function or Symbol 335
Select Line 335
Select Line Down 335
Select Line Up 335
Select Match 335
Select Next Window 336
Select Paragraph 336
Select Sentence 336
Select Symbol 336
Select To 336
Select To End Of File 336
Select To Top Of File 337
Select Word 337
Select Word Left 337
Select Word Right 337
Selection History 337
Setup Common Projects 338
Setup HTML Help 340
Setup WinHelp File 340
Show Clipboard 340
Show File Status 340
Simple Tab 341
Smart End of Line 341
Smart Beginning of Line 341
Smart Rename 341
Smart Tab 343
Sort Symbol Window 344

Sort Symbols By Line 344
Sort Symbols by Name 345
Sort Symbols By Type 345
Source Dynamics on the Web 345
Start Recording 345
Stop Recording 346
Style Properties 346
Symbol Info 350
Symbol Lookup Options 351
Symbol Window command 353
Symbol Window Properties 354
Sync File Windows 355
Synchronize Files 355
Syntax Decorations 356
Syntax Formatting 359
Tile Horizontal 363
Tile One Window 363
Tile Two Windows 363
Tile Vertical 363
Toggle Extend Mode 363
Toggle Insert Mode 364
Top of File 364
Top of Window 364
Touch All Files in Relation 364
Typing Options 364
Undo 367
Undo All 368
Vertical Scroll Bar 368
View Clip 368
View Relation Outline 368
View Relation Window Horizontal Graph 368
View Relation Window Vertical Graph 369
Window List 369
Word Left 370
Word Right 370
Zoom Window 370

Macro Language Guide 371

Overview 372
Basic Syntax Rules 372
Macro Functions 372
Running Macros 373
Statements 374
Variables 377
Declaring a Variable 377
Variable Name Expansion 377
Expanding Variables in a String 378

Variable Arithmetic 378
Indexing Into Strings 379
Record Variables 379
Record Variable Storage 380
Array Techniques 380
Special Constants 382
Operators 383
Conditions and Loops: if-else *and* while 384
The if Statement 384
The while statement 384
Break and Continue 385
Conditional Evaluation 385
Internal Macro Functions 386
String Functions 388
cat (a, b) 388
strlen (s) 388
isupper (ch) 388
islower (ch) 388
toupper (s) 388
tolower (s) 388
strmid (s, ichFirst, ichLim) 388
strtrunc (s, cch) 389
AsciiFromChar (ch) 389
CharFromAscii (ascii_code) 389
IsNumber (s) 389
User Input and Output Functions 390
Ask (prompt_string) 390
Msg (s) 390
StartMsg (s) 390
EndMsg () 390
Beep () 390
GetKey () 390
GetChar () 390
CharFromKey (key_code) 391
FuncFromKey (key_code) 391
IsFuncKey (key_code) 391
IsCtrlKeyDown (key_code) 391
IsAltKeyDown (key_code) 391
GetSysTime(fLocalTime) 392
CmdFromKey(key_value) 392
AssignKeyToCmd(key_value, cmd_name) 393
KeyFromChar(char, fCtrl, fShift, fAlt) 393
Buffer List Functions 395
BufListCount () 395
BufListItem (index) 395
File Buffer Functions 396
NewBuf (name) 396
OpenBuf (filename) 396

OpenMiscFile (filename) 396
CloseBuf (hbuf) 396
SaveBuf (hbuf) 396
SaveBufAs (hbuf, filename) 397
GetCurrentBuf () 397
SetCurrentBuf (hbuf) 397
GetBufHandle (filename) 397
ClearBuf (hbuf) 397
GetBufName (hbuf) 397
GetBufLineCount (hbuf) 397
GetBufLineLength (hbuf, ln) 398
GetBufLine (hbuf, ln) 398
PutBufLine (hbuf, ln, s) 398
InsBufLine (hbuf, ln, s) 398
AppendBufLine (hbuf, s) 398
DelBufLine (hbuf, ln) 398
CopyBufLine (hbuf, ln) 398
PasteBufLine (hbuf, ln) 398
SetBufIns (hbuf, ln, ich) 399
GetBufLnCur (hbuf) 399
GetBufSelText (hbuf) 399
SetBufSelText (hbuf, s) 399
MakeBufClip (hbuf, fClip) 399
GetBufProps (hbuf) 400
IsBufDirty (hbuf) 400
IsBufRW (hbuf) 401
SetBufDirty (hbuf, fDirty) 401
RenameBuf (hbuf, szNewName) 401
Environment Functions 402
PutEnv (env_name, value) 402
GetEnv (env_name) 402
SetReg (reg_key_name, value) 402
GetReg (reg_key_name) 402
IsCmdEnabled (cmd_name) 402
RunCmd (cmd_name) 403
Window List Functions 404
WndListCount () 404
WndListItem (index) 404
Window Functions 405
NewWnd (hbuf) 405
CloseWnd (hwnd) 405
SetCurrentWnd (hwnd) 405
GetCurrentWnd () 405
GetWndHandle (hbuf) 405
GetWndBuf (hwnd) 406
GetWndDim (hwnd) 406
GetWndLineWidth (hwnd, ln, cch) 406

GetNextWnd (hwnd) 407
GetWndSel (hwnd) 407
SetWndSel (hwnd, selection_record) 408
GetWndSelLnFirst (hwnd) 408
GetWndSelLnLast (hwnd) 408
GetWndSelIchFirst (hwnd) 409
GetWndSelIchLim (hwnd) 409
MaximizeWnd (hwnd) 409
MinimizeWnd (hwnd) 409
ToggleWndMax (hwnd) 409
IsWndMax (hwnd) 409
IsWndMin (hwnd) 409
IsWndRestored (hwnd) 409
ScrollWndToLine (hwnd, ln) 409
ScrollWndVert (hwnd, line_count) 410
ScrollWndHoriz (hwnd, pixel_count) 410
GetWndVertScroll (hwnd) 410
GetWndHorizScroll (hwnd) 410
GetWndLineCount (hwnd) 410
Bookmark Functions 411
Bookmark Record 411
BookmarksCount () 411
BookmarksItem (index) 411
BookmarksAdd (name, filename, ln, ich) 412
BookmarksDelete (name) 412
BookmarksLookupName (name) 412
BookmarksLookupLine (filename, ln) 412
Symbol List Functions 413
SymListCount () 413
SymListItem (hsyml, isym) 413
SymListInsert (hsyml, isym, symbolNew) 413
SymListRemove (hsyml, isym) 414
SymListNew () 414
SymListFree (hsyml) 414
Symbol Functions 415
Symbol Record 415
SymbolChildren (symbol) 416
SymbolParent (symbol) 417
SymbolStructureType (symbol) 417
SymbolDeclaredType (symbol) 417
SymbolLeafName (symbol) 417
SymbolContainerName (symbol) 418
SymbolRootContainer (symbol) 418
GetSymbolLocation (symbol_name) 418
GetSymbolLocationEx (symbol_name, output_buffer, fMatchCase, fLocateFiles, fLocateSymbols) 419
GetSymbolLocationFromLn(hbuf, ln) 420
JumpToLocation (location_record) 420

GetCurSymbol () 420
GetSymbolLine (symbol_name) 421
JumpToSymbolDef (symbol_name) 421
GetBufSymCount(hbuf) 421
GetBufSymLocation(hbuf, isym) 421
GetBufSymName(hbuf, isym) 421
Searching Functions 423
SearchForRefs (hbuf, word, fTouchFiles) 423
GetSourceLink (hbufSource, lnSource) 423
SetSourceLink (hbufSource, lnSource, target_file, lnTarget) 424
SearchInBuf (hbuf, pattern, lnStart, ichStart, fMatchCase, fRegExp, fWholeWordsOnly) 424
LoadSearchPattern (pattern, fMatchCase, fRegExp, fWholeWordsOnly) 425
ReplaceInBuf (hbuf, oldPattern, newPattern, lnStart, lnLim, fMatchCase, fRegExp, fWholeWordsOnly, fConfirm) 425
Project Functions 426
AddFileToProj(hprj, filename) 426
RemoveFileFromProj(hprj, filename) 426
NewProj (proj_name) 426
OpenProj (proj_name) 426
CloseProj (hprj) 426
DeleteProj (proj_name) 426
GetProjName (hprj) 426
GetProjDir (hprj) 427
GetCurrentProj () 427
GetProjFileCount (hprj) 427
GetProjFileName (hprj, ifile) 427
GetProjSymCount (hprj) 427
GetProjSymName (hprj, isym) 427
GetProjSymLocation (hprj, isym) 428
SyncProj (hprj) 428
SyncProjEx (hprj, fAddNewFiles, fForceAll, fSupressWarnings) 428
Miscellaneous Macro Functions 430
DumpMacroState (hbufOutput) 430
GetProgramInfo () 430
GetProgramEnvironmentInfo () 431
Other Information about Macros 433

Appendix: Upgrading From Version 2 434

Upgrading to Version 3 from Version 2 435
Installing Version 3 435
Opening Older Projects 435
Loading Old Customizations 435
Using Version 3 and Version 2 Together 436
New Commands 437
New Command List 437
File Format Compatibility with Older Versions 441

Index **443**

Introduction

Thank you for purchasing Source Insight. Source Insight has been continuously improved for over ten years. We believe that Source Insight will improve your productivity and increase your enjoyment of programming.

This chapter introduces you to some of Source Insight's capabilities.

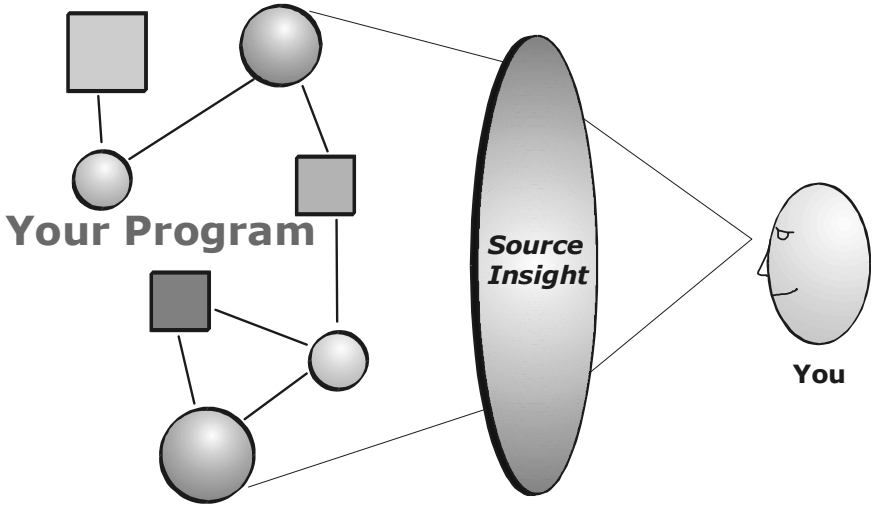
The Big Picture

The Problem

You have a multitude of source files spread out all over the place. You have to deal with functions that somebody else wrote. You have to figure out how some piece of code works and see all of its clients. You didn't write the code, or you wrote it in a past life.

You may be one of the cleverest developers in the world, but if you can't find all the myriad pieces of your program, or can't get your head wrapped around the code, then you will not be very productive.

Enter Source Insight.



The Solution

Source Insight was designed to enhance your ability to understand and modify your program. Our company mission is to increase programming team productivity by clarifying source code, presenting information in a useful way, and allowing programmers to modify software in large, complex projects.

Think of your program's source code as a free form database of information. It has not only classes, members, and functions in it,

but it has many important comments. (You *do* have comments, don't you?)

Your source code also has a history. In fact, many large programs have a long lifetime that includes contributions by many programmers over many years. Some of it is not pretty, but you have to live with it.

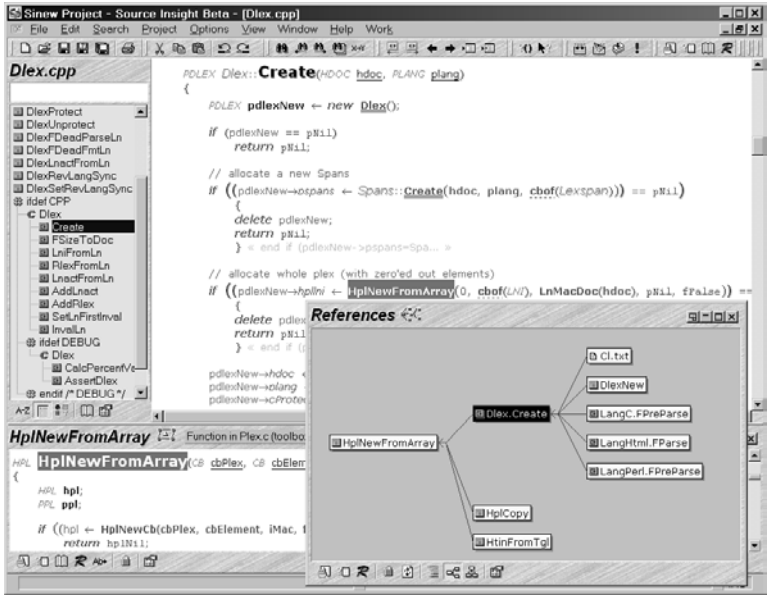
Source Insight acts as an information server that surrounds your project's source code. With it, you can have instant access to symbolic and textual information in your program.

Whether you are new to a project, or an old-timer, Source Insight will give you useful leverage to stay productive.

Feature Highlights

Source Insight™ is a project-oriented program editor and program browser designed for large programming projects under the Microsoft® Windows™ family of operating systems. It is especially helpful for large C/C++ and Java programs, and projects that involve multiple languages and multiple programmers.

Source Insight features quick and innovative access to source code and source information. Unlike many other editor products, Source Insight parses your source code and gives you useful information and analysis right away, while you edit.



Source Insight uses innovative syntax formatting and dynamic relationship graphing to illuminate source code design.

Up-To-Date Information

Because programs are constantly under development, it's important that even symbols in code that will not compile can be browsed in with up-to-the-second accuracy. Source Insight automatically builds and maintains its own high performance symbol database of functions, methods, global variables, structures, classes, and other types of symbols defined in your project source files. Source Insight maintains its symbol database to provide browsing features instantly, without having to compile the project or having to depend

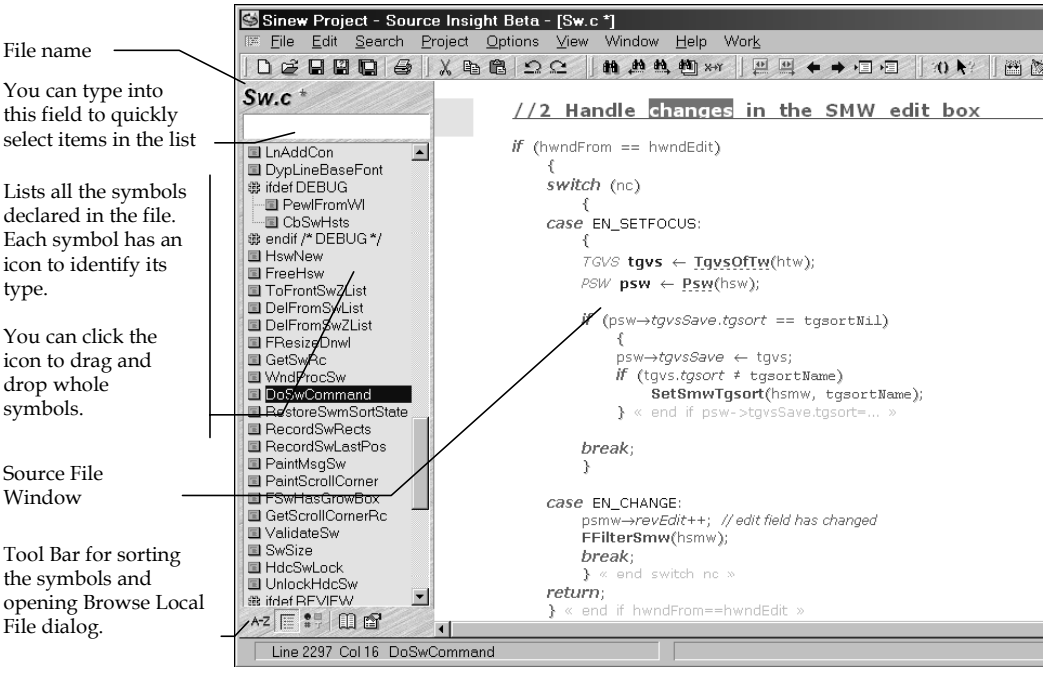
on the compiler to provide browser files. Source Insight quickly and un-intrusively updates its information about your files, even while you edit code. Furthermore, the symbol features are built into each Source Insight project automatically. You don't need to build any extra tag files.

Dynamic Type Resolution

Source Insight decodes the types of variables, including class inheritance, dynamically while you edit. Source Insight's knowledge of classes gives you accurate information as soon as you need it.

Symbol Windows

Symbol Windows appear on the side of each source window and are dynamically updated to allow easy navigation within each file and to provide a quick overview of the file.



You can click on any symbol in the Symbol Window and quickly jump there. You can also drag and drop symbols in to rearrange your code. The Symbol Window can be sorted by name, line number, and type. You can activate the Symbol Window and type the first few letters of a symbol's name in order to quickly move to it. The Symbol Window also displays `#ifdef-#endif` nesting

levels and symbol type icons for quick identification and orientation.

Dynamic Context Window

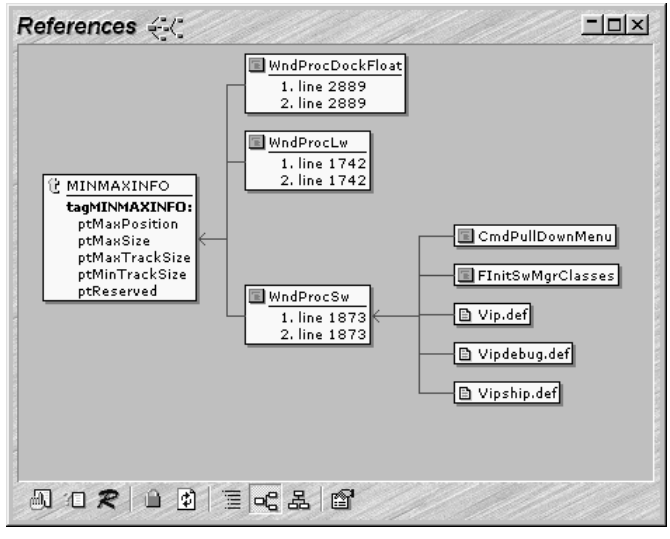
Source Insight 2.0 introduced an innovative feature called the Context Window. The Context Window automatically displays symbol definitions based on what identifier your cursor is in, or on what you are typing.

The Context Window updates in the background and tracks what you are doing. You can click on an identifier, and the Context Window will automatically show the symbol's definition. If the identifier is a variable, the Context Window will decode its declaration to show you its base structure or class type.

The Context Window also will automatically display files selected in the Project Window, symbols in the Relation Window, and clips selected in the Clip Window.

Dynamic Relation Window

The Relation Window is a Source Insight innovation that shows interesting relationships between symbols. It runs in the background and tracks what symbols you have selected. With it, you can view class hierarchies, call trees, reference trees, and more. The beauty of the Relation Window is that you don't have to do anything special. It works in the background while you work, but you can interact with it when you want to. You can also have several Relation Windows open, each showing different types of information.

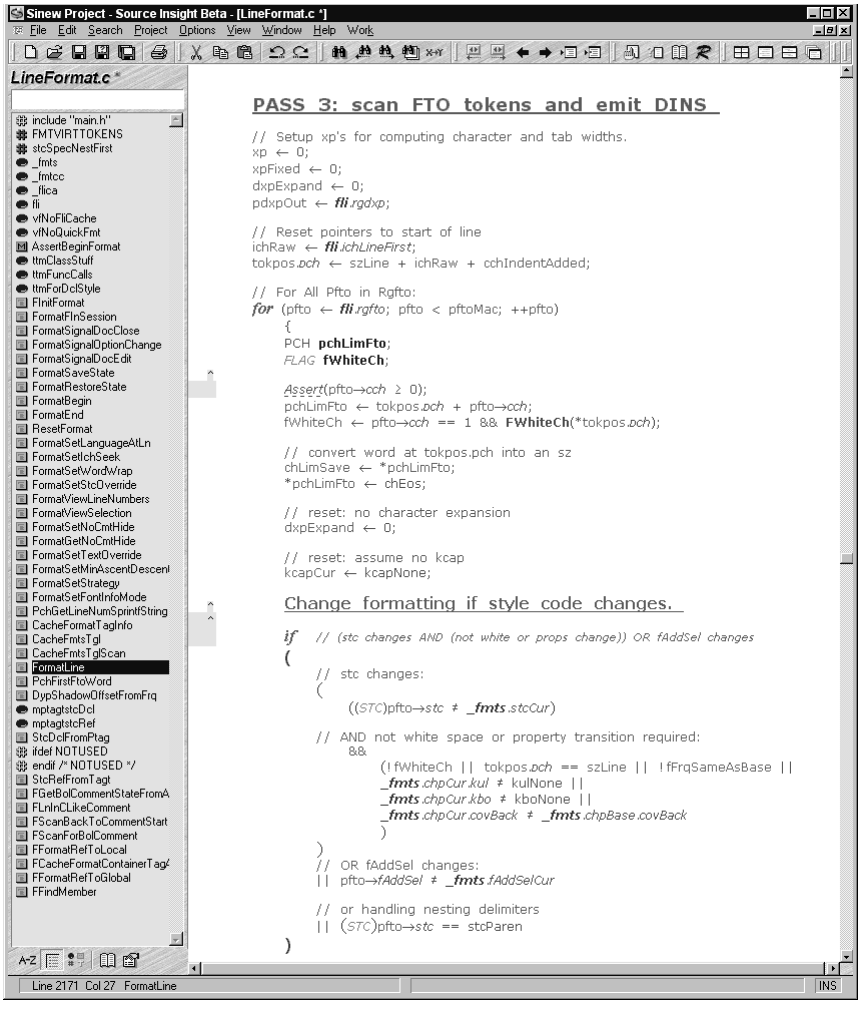


The Relation Window is showing references to a type, and indirect references through a function.

The Relation Window can be viewed either graphically, or in outline format.

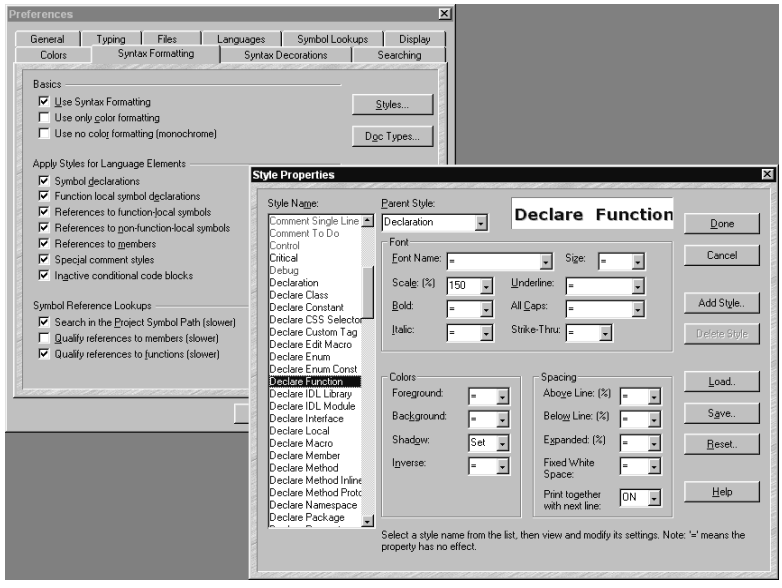
Syntax Formatting

Syntax Formatting is an important Source Insight innovation that renders information in a dense, yet pleasing and useful way. It provides vastly improved display capabilities, including full rich text formatting with user-defined styles. Source Insight applies styles automatically based on lexical and symbolic information about your project.



Syntax Formatting makes your code come to life! This example demonstrates Source Insight's unique comment heading styles, scaled nested parentheses, and a variety of symbol reference styles. The yellow marks in the left margin indicate lines that have been edited.

Syntax Formatting adds valuable information while you read your code. For example, references to local variables can look different from references to global variables. Or, references to functions can look different from references to C function-like macros. With Syntax Formatting, it becomes instantly obvious what an identifier refers to, or if it is misspelled.



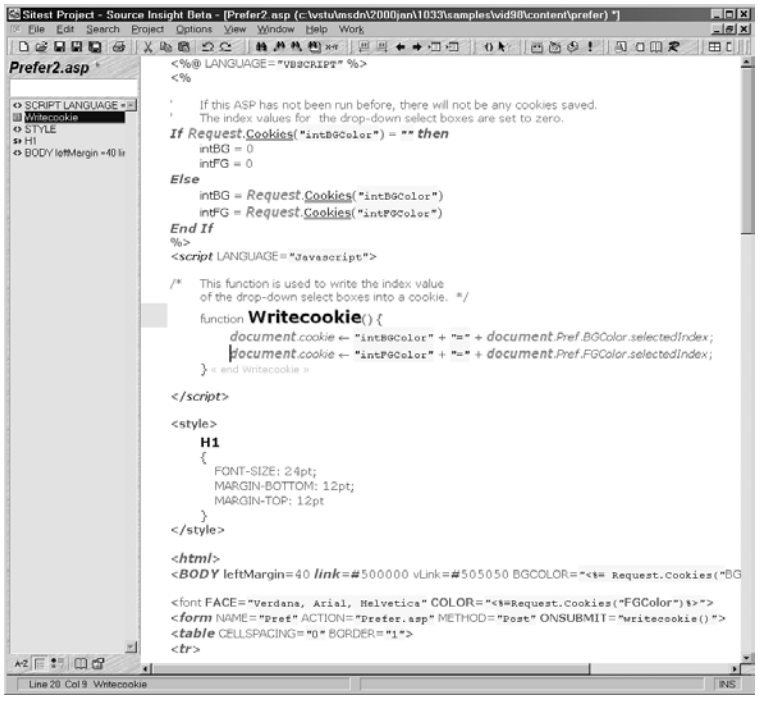
The Style Properties dialog box allows users to edit many formatting options. The Preferences dialog box allows users to control what language elements should have styles applied to them.

Context-Sensitive Smart Rename

Source Insight's indexes allow you to effortlessly rename variables, functions, and other identifiers in one simple step. Source Insight's context-sensitive smart matching feature is smart enough to rename local scope variables, as well as global or class scope identifiers.

Mixed Language Editing

Source Insight supports HTML and Active Server Page files (ASP and JSP) with embedded script. The embedded script can be browsed symbolically, and displays with appropriate syntax formatting.



Embedded script in ASP and HTML is also shown with syntax formatting.

Keyword Searches

Source Insight's Search Project command allows keyword style searching, similar to an Internet search engine. This lets you find sections of code that refer to one or more topics within a specified number of lines. For example, you might type "save disk (copy or duplicate)" and Source Insight will find all references to the words "save", "disk", and either "copy" or "duplicate" that occur near each other (as well as word variations, such as "saves", "saved", and "saving").

Symbolic Auto-Completion

When you begin to type an identifier name, Source Insight will pop up a list of potential identifier names. Source Insight can show you function and variable names, as well as class and structure fields up to many levels of depth. Source Insight decodes the types of variables (including inheritance) on the fly.

Quick Access to All Symbols and Files

Source Insight provides more useful programming information than any other editor. Not only are symbol definitions tracked dynamically, but also information is offered in its most useful form.

With Source Insight, you can surf your project the way you would a web site. You can just double click on a local or global symbol, and Source Insight takes you to the definition, or can pop up a quick information window. You can click on a symbol, and within seconds, have a list of all references to that symbol anywhere in the project. A symbol-browsing dialog box allows you to perform regular expression searches to locate symbols.

Programming Project Support

Entire source directory trees, even multiple directories across your network, can be added to a Source Insight Project. You can specify a file name quickly without having to know what directory it is in. Source Insight automatically maintains its own database of symbols and indexes for each project. Project reports and cross-references can also be generated. When your source control program updates files in the project, Source Insight notices and incrementally updates the symbol information for you automatically.

Team Programming Support

Changes made by any member of a programming team are reflected automatically because the entire code base is scanned and resynchronized as needed.

Programmers need not be concerned with the organization of the project and its files, because they can instantly jump to the definition or usages of any symbol, and can access modules and other symbols without having to know what directory, machine, or file they are in.

Source Insight gives each programmer the leverage to easily understand and edit large, detailed projects created by groups of programmers.

Find References Quickly

Source Insight's symbol indexes allow you to locate references to symbols across the project in seconds, and create a listing containing active source links to all locations.

Source Links

Source Links let you jump between interesting places instantly. Source Links are hypertext-like links that connect a location in one

file with a location in another file. Source Links are used to link search results with matches, and to link compiler errors with their targets. You can also parse file specifications out of any file and create links to those files. Source Links are actively maintained as files are edited. You can insert text anywhere in a file and the links will be retained on the correct lines.

Fast Project-Wide Search and Replace

Source Insight can quickly search and replace in project files. The result of each search is added to a Search Results window, which contains active source links to all the search matches. Source Insight's search index makes project-wide searches take only a few seconds. Regular expression search patterns are supported.

Project Window

Source Insight's Project Window displays the contents of your project, and it has several modes. In the File List mode, it lists all files in the current project. You can quickly open any file in the project from the Project Window, or drag files from the Windows Explorer onto the Project Window to add files to the project.

The Project Window also has modes to display symbols by category, files by category, and all project symbols in a flat list. You can use partial matching on names or parts of names to quickly locate items.

Source Insight can handle projects with millions of lines of code and hundreds of thousands of declared symbols.

Integrates with External Compilers and Tools

Source Insight integrates with external tools, such as compilers, make programs, filters, and source control programs by using Custom Commands. Projects can be compiled from inside Source Insight and compiler errors are tracked automatically while you edit. External tools can be launched concurrently in a shell command window from within Source Insight. Program output can be redirected to a file buffer or can be parsed for errors messages. You can add your own Custom Commands, which spawn external tools.

Clip Window

You can easily rearrange code and insert boilerplate text by using the Clip Window. The Clip Window contains clips of text that you can keep handy for dropping into their source files when needed. Clips are automatically saved and maintained across sessions. Clips also remember what function or symbol they came from.

Full Featured Editor

Of course, Source Insight offers great editing features, such as multi-level Undo & Redo per file, smart indenting, syntax coloring, parentheses and brace matching, renumbering, keystroke and command recording, and special selection modes for selecting blocks, functions, paragraphs, and whole words. The mouse is fully supported. Multiple windows can be open on the same file. Workspaces are used to restore files and windows from previous sessions.

Drag and Drop Editing

Source Insight supports drag and drop editing of text between source files and between clips in the Clip Window. Whole symbols can be dragged and dropped from the Symbol Window, which makes rearranging functions and things very easy. You can also drop files on Source Insight's Project Window to add a file to a project, or onto the Clip Window to load a new clip.

Two-Stage Line Revision Marks and Selective Line Restoration

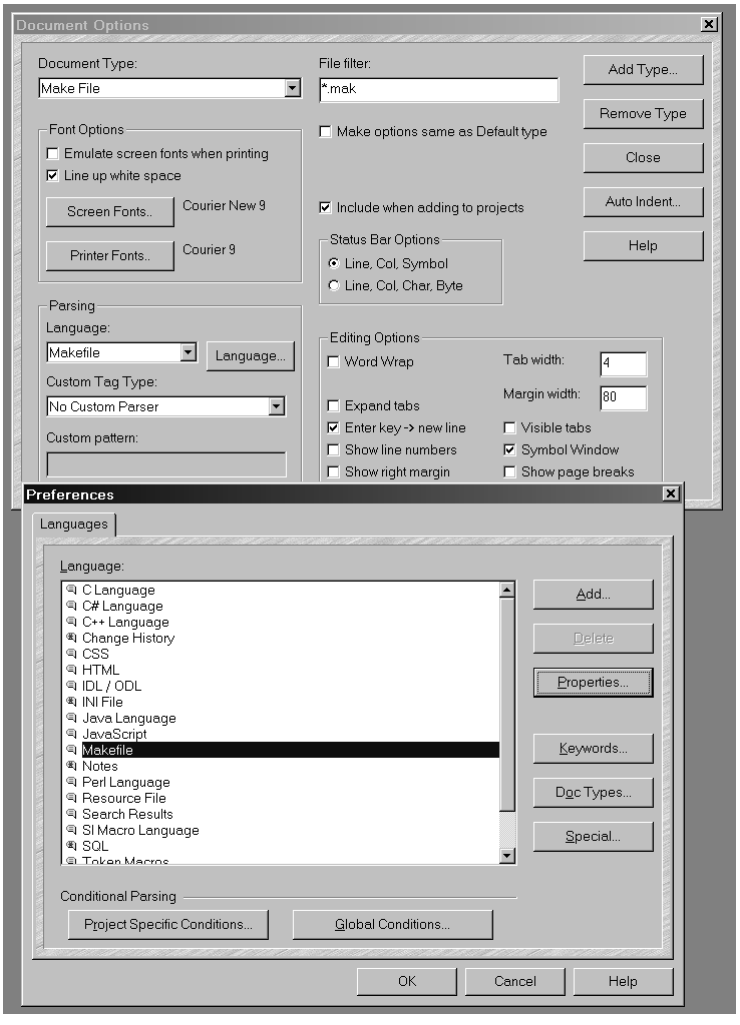
Source Insight displays line revision marks in the margin next to lines that have been changed, or where lines have been deleted. This makes it easy to see where you have made changes in your files. Not only can you see where you made changes, but also you can restore them to their original text with the Restore Line command. The Restore Line command is undoable. This gives you powerful, out-of-order undo capabilities!

The undo and change history for each file is preserved after you save the file. The line revision marks also change color when a file is saved. After saving a file, you can still see what lines were edited, and restore them, or perform undo operations.

Extensible Document Types and Languages

You can teach Source Insight about new file types by defining Document Types. This allows different editing, display, and language parsing options for different types of files.

You can also add your own custom languages to Source Insight. A custom language specifies syntax rules, syntax formatting keywords, and simple parsing expressions.



Files are mapped to Document Types; Document Types are mapped to a Language. You can add new Document Types, and Languages.

Crash Recovery Offers Full-Time Protection

Source Insight saves your editing changes transparently and frequently to a recovery file. In the event of a computer crash, Source Insight can recover all the changes made to files, even if you didn't save them. This is not an auto-save feature, which interrupts you so that files can be saved. Only the changes you have made are stored in the recovery file and only when you are idle. You can specify how often the recovery file should be saved.

Persistent Workspaces

You can group sets of files and other session information into Workspaces. You can save all session state into workspace files, and restore sessions from other workspace files easily. Source Insight saves the current workspace automatically when you exit. The workspace is restored when you run Source Insight again, or when you open other projects. You can exit Source Insight, close a project, or even crash your machine, but *everything* will be just as you left it when you start it up again.

Customizable Menus and Keyboard

Not only is the keyboard configurable, but also the mouse buttons and the menus are fully configurable. All your configuration settings can be stored and restored from configuration files very easily, while running Source Insight. There is no need to write custom macros or to use a separate "setup" program.

Windows 2000 Support

Source Insight supports Windows 2000 features, such as semi-transparent windows for floating tool windows, and Terminal Server sessions.

Outstanding Windows User Interface

Source Insight is a full 32 Bit implementation and supports Multiple Instances, Long and UNC File Name, Right-Click Shortcut Menus, and Toolbars.

Source Insight uses right mouse button shortcut menus in many of its windows to provide easy access to commands and object properties.

Many useful windows can be either floating or docked to the main application window for flexibility.

Real World Tested

Source Insight is an industrial strength editor that is used by thousands of programmers at major public software companies today. Source Insight has been used on complex commercial projects containing many thousands of files, many millions of lines of code, and hundreds of thousands of declared symbols.

Speed and Convenience

The philosophy of Source Insight is to increase programming team productivity by clarifying source code, presenting information in a

useful way, and allowing programmers to modify software in large, complex projects. By quickly providing you with complete project-wide program information, and giving you a rich program-editing environment, Source Insight will enable you to work quickly and smartly.

System Requirements:

Operating Systems:

- Windows XP
- Windows 2000 Professional or Server
- Windows NT 4.0 SP3
- Windows Me
- Windows 98
- Windows 95 with Internet Explorer 4.0+

Machine: Pentium or faster, Pentium II or better recommended.

Memory: 64 MB, 128 MB or more recommended.

Disk Storage: 4 MB.

What's New in Version 3

A lot has changed in Source Insight since version 2.x. Here is a summary of the largest changes.

Improved Language Features

Language specific features, such as parsing and symbol lookup, have changed significantly in version 3.0. Some of the changes include:

- The context-sensitive symbol lookup engine can track class and struct members, and decode class inheritance dynamically without compiling. Version 2.1 does not track field members or inheritance.
- Supports nested C++ and Java classes and structures.
- Supports C++ class and function templates.
- Supports C++ namespaces.
- Supports interpolated C structs and unions (i.e. inlined structures).
- Supports C++ template classes and functions.
- Supports anonymous structs and unions.
- User definable token substitution macros can replace source code tokens upstream of Source Insight's parsers so it can handle variations in keywords and preprocessor usage.
- User definable compile-time constants used for marking #ifdef branches active or inactive.
- Code metrics.
- New built-in language parsing and display support:
    - Perl and PerlScript
    - Visual Basic and VBScript
    - JavaScript and JScript
    - HTML, ASP, and JSP with embedded script.
    - C# (C Sharp)
- User-defined custom language support.

Improved Browsing and Analysis Features

Source Insight continues to excel at proving first class access to symbolic information in your programs.

- New Relation Window shows dynamic call trees, class trees, and reference trees that update while you work.
- Improved Project Window with better file and symbol listing features.
- Symbol Syllable Indexing – indexes sub strings of symbol names so you can type their partial names quickly without knowing exactly what the symbol name begins with.
- Optimized symbol database for supporting even larger projects more quickly.
- Improved "Lookup References" command uses smart reference matching to show only the appropriate references to symbols. It can also skip or include comments or inactive blocks of code.
- New Search Project command features keyword searching – similar to an Internet search, which can find words near each other in your project.

Improved Editing and Display Features

Editing and display features are greatly improved in version 3.0.

- Syntax Formatting provides a vastly improved display capability, including full rich text formatting with user defined styles. Source Insight applies styles automatically based on lexical information. Syntax Formatting includes:
  - Function, class, and variable declaration styles, in addition to others.
  - Several rich "comment" styles.
  - Styles for references to different types of symbols such as locals, parameters, globals, macros, functions, etc.
  - Auto-annotations and special Syntax Decorations.
- Auto symbol name completion while you type.
- Incremental search.
- New context-sensitive Smart Rename command renames a symbol in all the appropriate contexts. You can rename local variables instantly.
- Saving a file preserves its undo and change history, and displays two-stage revision marks.
- Line number display.
- Visible page-breaks and page numbers.
- Visible right-margin.
- User defined "Work" menu.
- Improved file handling during save operations.
- Miscellaneous improvements to the UI, including:
  - Better window docking.
  - Customizable fonts and colors for all windows.
  - Dialog box position memory.
  - Multiple selection lists.
  - Improved toolbars.

Setup and Quick Start

This chapter describes how to install Source Insight and run it for the first time. After setting up Source Insight, we will take a quick tour.

Installing Source Insight

Installing Source Insight is very simple and straightforward. For the most part, Source Insight is self-configuring.

Installing on Windows NT/2000/XP

Make sure you have system permissions to install software on your machine.

In order to install Source Insight on Windows NT-based systems, you will need to have system permissions to install software, and to modify the HKEY_LOCAL_MACHINE registry hive. However, once Source Insight is installed, it can be run from almost any user account. Source Insight keeps separate preferences settings for each user.

Insert the CD-ROM

To install Source Insight, insert the Source Insight CD-ROM in your CD or DVD ROM drive. The setup program will start automatically. If you have the auto-run feature disabled on your machine, then you will need to run Setup.exe found in the root folder of the CD-ROM.

Source Insight is normally distributed on CD-ROM. If you require it on a different media format, please contact Source Dynamics at support@sourceinsight.com.

Choosing a Drive for the Installation

The setup program will ask you for the name of a directory to install Source Insight in. The setup program will propose C:\Program Files\Insight3 by default.

Note: If you want to continue to use version 2.x on your machine, you should install version 3.1 into a *different* directory than version 2.x.

Source Insight requires approximately 4 MB of hard disk space for a complete installation.

Note: Do **not** install Source Insight on a remote **network drive**. Source Insight's setup program will modify *your* registry on your own machine. An installation on a network drive will not work correctly for anyone else.

Upgrading from Version 2

If you already have Source Insight 2.x installed on your machine, please refer to "Appendix: Upgrading From Version 2" on page 434.

Using Version 3 and Version 2 Together

You can keep older versions of Source Insight installed.

You can use both version 3.1 and version 2.x together on the same machine. They each use separate registry settings and should not conflict. However, you should follow these guidelines:

- If you currently have version 2.x installed on your machine, you should install version 3.1 into a *different* directory than version 2.x
- Don't run instances of version 2.x and version 3.1 at the same time.
- Creating separate projects for version 3.1 is recommended, although the project files are somewhat upward and downward compatible.
- You can open a version 2.x project, but you will have to click the Browse button in the Open Project dialog box to locate the old .PR file yourself. If you installed version 3.1 in a new directory (recommended), then version 3.1 will have no foreknowledge of the old projects already created.
- You can open your old configuration file with the Options > Load Configuration command. Point to your old 2.x directory and your old *.CF file. Note that new configuration files have a .CF3 file extension.

Configuring Source Insight

A Source Insight item will be added to the Programs menu of the Start menu.

After the Setup program finishes copying files to your hard disk, Source. You will be asked to type your name and organization in the sign-on window. This window only appears this one time.

Entering Your Serial Number

For permanent use, you will need to purchase a valid serial number.

Source Insight asks you to enter a serial number when you run it. If you are only evaluating Source Insight, you do not need a serial number. Source Insight will run without a serial number for 30 days. Click the Try button to continue evaluating Source Insight without a serial number.

If you have paid for a Source Insight license, then you should enter your serial number when prompted. Once a valid serial number is entered, Source Insight's evaluation time limit is removed, and it will be "unlocked" indefinitely.

Creating Common Projects

Source Insight will ask you at this point if you want to create common projects.

Common Projects are external projects that contain commonly used declarations.

In order for Source Insight to provide symbol completion, and other symbolic features for standard libraries, such as the C Runtime, or Java standard packages, you need to setup separate projects for those libraries. Source Insight will resort to searching these projects if a symbol cannot be found in your current project.

For each common project, you are asked to locate the directory where the corresponding files are located on your disk. If you installed the source code for your libraries on your disk, then you can take advantage of Source Insight to use the source code as a basis for the projects. For instance, you might click on a call to the function `strtok in the C Runtime Library`, and Source Insight will locate the source code for `strtok`.

The projects that you create at this point are automatically added to the project symbol path. Later, if you want to change that path, run the Preferences: Symbol Lookups command (Options menu) to edit it. You can always create projects later and add them to the path.

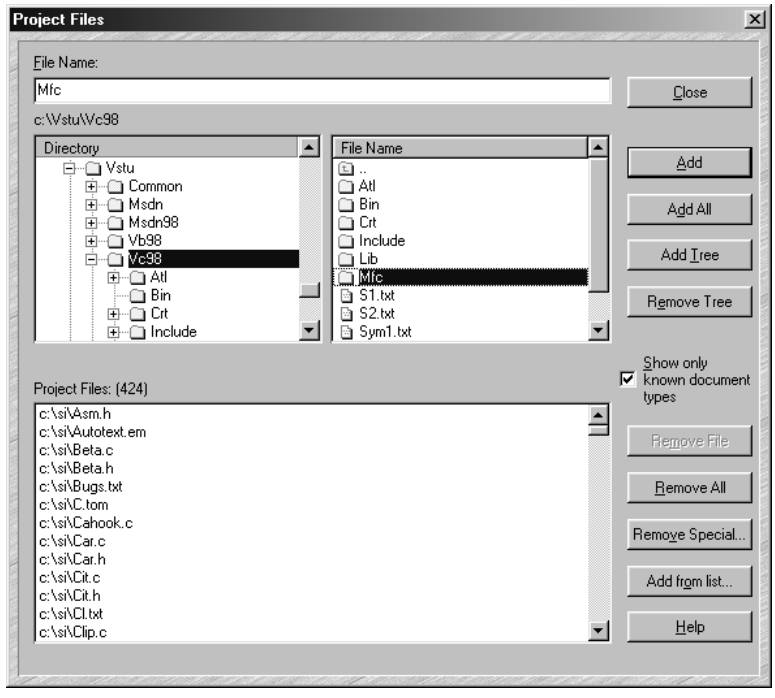
See "Setup Common Projects" on page 338.

Creating a Project

A project is a collection of source files.

Source Insight is built around projects. A project is a collection of source files. When you create a project, you need to add files to it. Source Insight records what files are in the project by keeping a simple file database for the project.

The Add and Remove Project Files dialog box lets you add individual files, or whole directory trees to your project. For more information, see "Add and Remove Project Files" on page 174.



As you create new files, they can be added to your project when you save them. If new files appear in your project directory or subdirectories, they can also be added automatically to your project by running the Synchronize Files command.

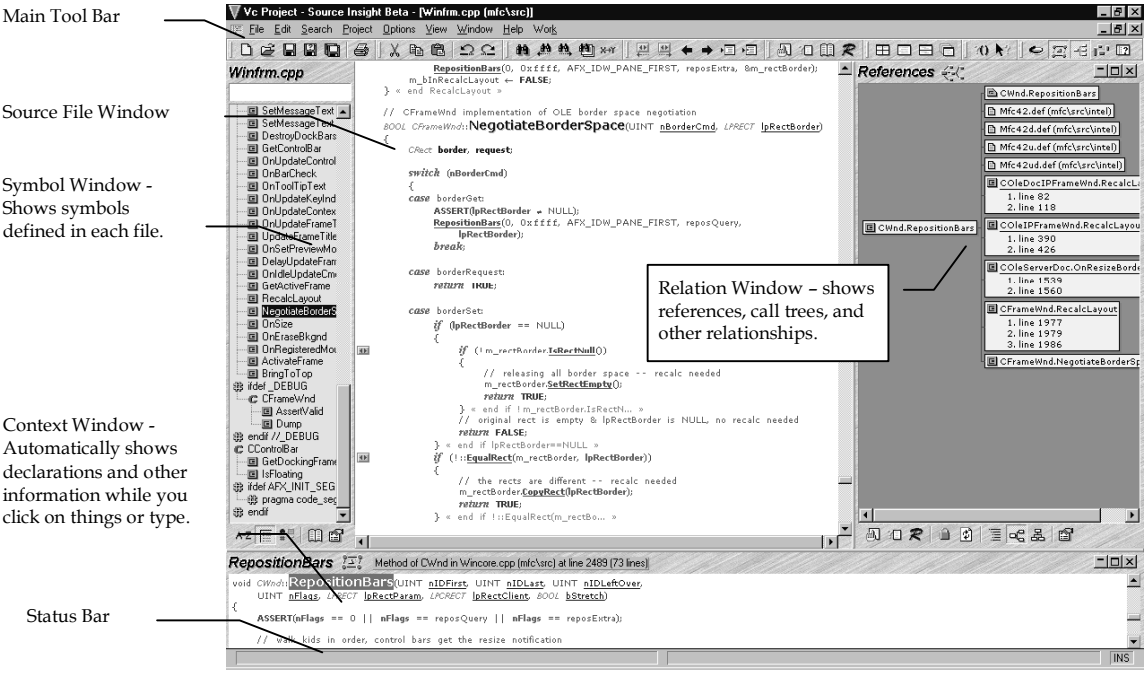
Where to Go Next?

Please continue to the Window Tour chapter, to get useful overviews of the different windows in Source Insight.

Window Tour

This chapter describes the different windows available in Source Insight.

Main Source Insight Window


Main Tool Bar
Source File Window
Symbol Window - Shows symbols defined in each file.
Context Window - Automatically shows declarations and other information while you click on things or type.
Status Bar
Relation Window – shows references, call trees, and other relationships.
Vc Project - Source Insight Beta - [Winfrm.cpp (mfc\src)]
File Edit Search Project Options View Window Help Work
Winfrm.cpp
References
RepositionBars
Method of CWnd in Wincore.cpp (mfc\src) at line 2489 (73 lines)
INS

The main Source Insight program window, showing a source file window with a symbol window attached on the left side, and a Relation Window docked to the right edge. The Context Window appears docked at the bottom edge.

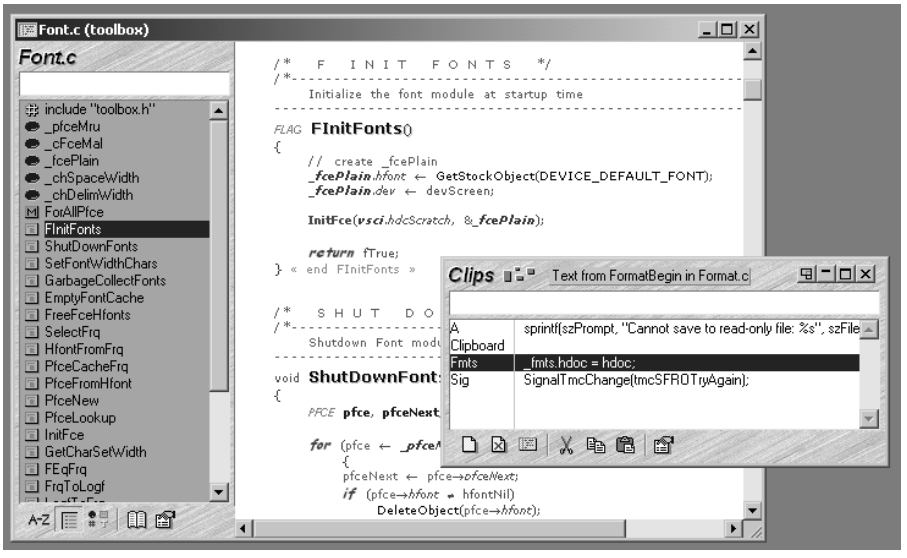
Source Insight is an MDI (Multiple Document Interface) application. This means that each source file you open has its own child window contained within the Source Insight application window.

The main Source Insight application window contains the main toolbar at the top.

Floating Windows

Floating windows can be docked to an edge of the main Source Insight window.

Floating windows, such as the Context Window and Relation Window, will float as separate windows, always in front of the main window. By dragging a floating window to an edge of the main window, you can dock it to the main window.



The Clip Window is a floating window that stays in front of the main Source Insight window.

Tip: To get a floating window to dock where you want it, drag it so that the mouse cursor itself is near the edge where you want to dock the window.

Transparent Floating Windows

Source Insight supports semi-transparent floating windows. This makes the floating windows a little like a HUD (Heads Up Display) in a game.

Source Insight supports semi-transparent floating windows on Windows 2000.

If you are using Windows 2000 or newer, which supports translucent window modes, then the floating windows will also have a button to toggle the window's transparency.

When a window is transparent, you can also click through it to the text below it, as long as you are not clicking inside an object within the floating window.

Toolbars

The Main toolbar appears at the top of the Source Insight program window. You can toggle the whole Main toolbar on and off with the View > Toolbars > Main Toolbar command.

The View > Toolbar menu item controls what toolbars are visible.

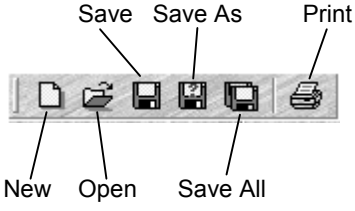
The Main toolbar is made up of smaller sub toolbars. Each sub toolbar can be displayed independently using the View > Toolbars menu. You can also drag the sub toolbars around within the Main toolbar.

The position of each toolbar is saved in the configuration file automatically.

Each toolbar icon corresponds to a Source Insight command. Please refer to the Command Reference chapter for information on each command.

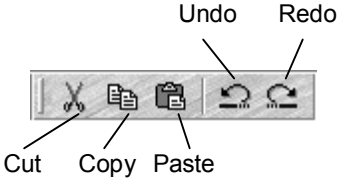
Standard Toolbar

The Standard toolbar contains the basic file operations.



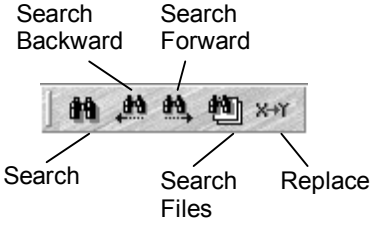
Edit Toolbar

The Edit toolbar contains the basic editing operations, like cut, copy, and paste.



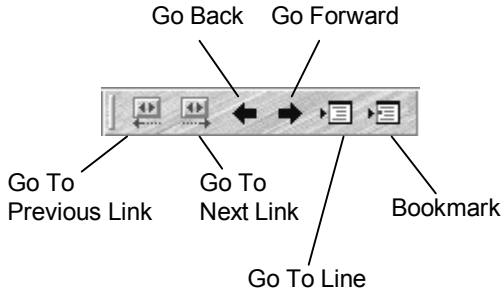
Search Toolbar

The Search toolbar contains searching commands.



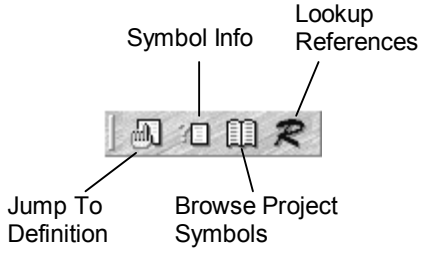
Navigation Toolbar

The Navigation toolbar contains commands for moving around in, and in-between files.



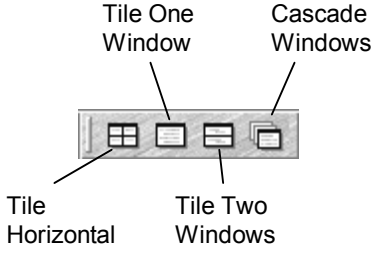
Symbols Toolbar

The Symbols toolbar contains commands for accessing symbolic information.



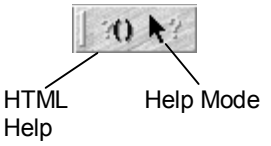
Window Toolbar

The Window toolbar contains commands for arranging windows.



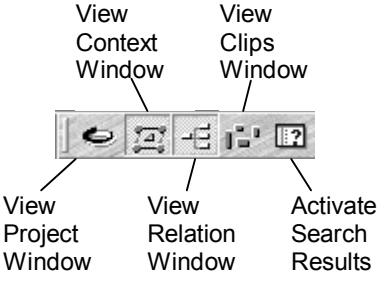
Help Toolbar

The Help toolbar contains commands for accessing on-line help.



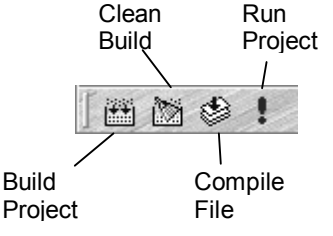
View Toolbar

The View toolbar contains commands for showing and hiding the auxiliary windows, like the Context Window and Project Window.



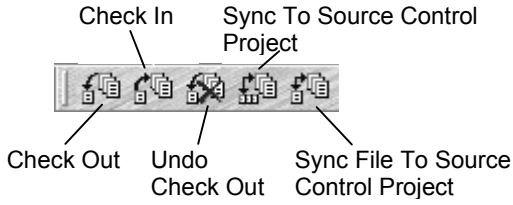
Build Toolbar

The Build toolbar contains commands that are used typically to build your project's executable. These commands are defined as Custom Commands.



Source Control Toolbar

The Source Control toolbar contains commands that are used to access your source control (also known as version control) system. These commands are defined as Custom Commands.

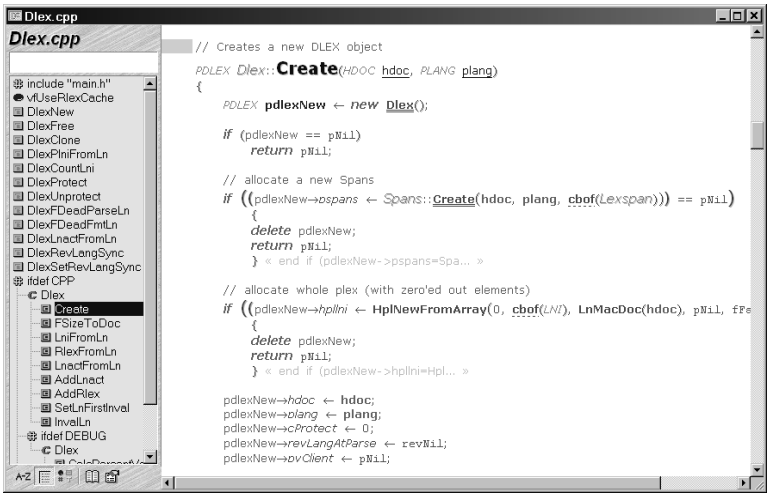

Check In
Sync To Source Control Project
Check Out
Undo Check Out
Sync File To Source Control Project

Source File Window

When you open a source file, it appears in its own source file window. This window is where you do all of your regular editing. A source file window is an MDI window.

Source file windows display file buffers.

When you open a file that has a language attached, a Symbol Window will be attached to the left side of the source file window. You can control whether a Symbol Window is used by selecting Document Options and setting the **Use symbol window** check box accordingly.



A source file window, which has a symbol window attached to the left side.

Symbol Window

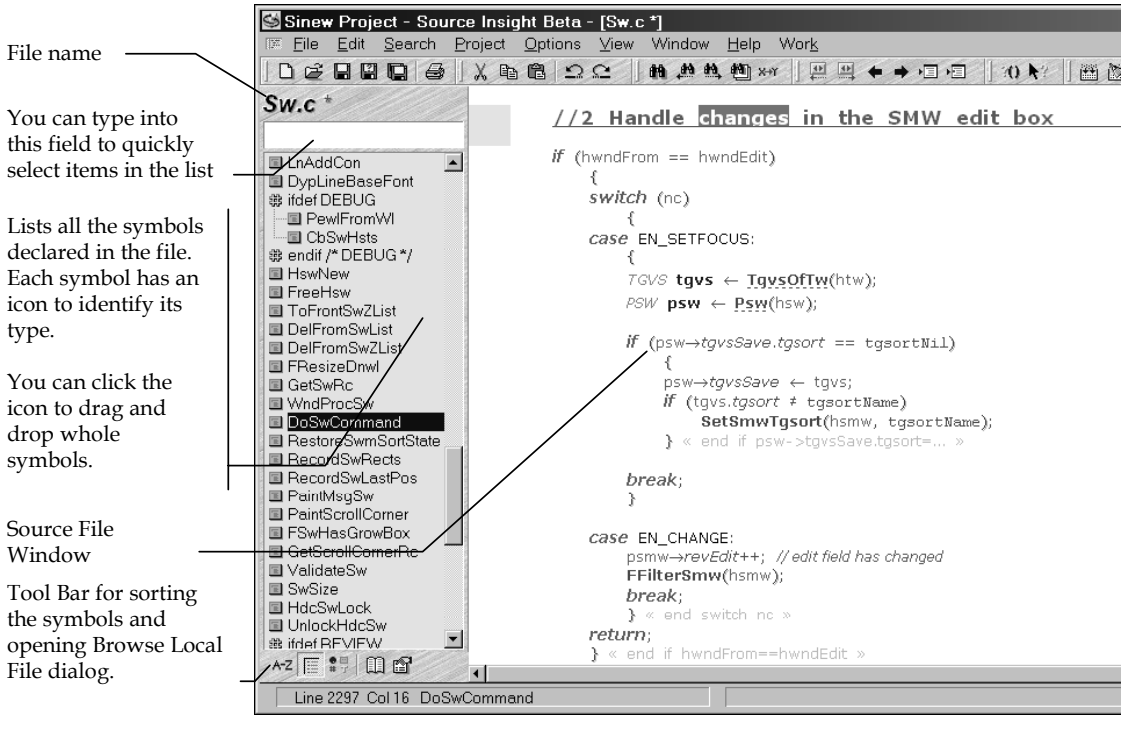
Symbol Windows are usually attached to the side of each source file window.

Symbol Windows appear at the left side of each source file window that has a language specified. The Symbol Window lists all the symbols defined in the file. For example, all functions, structs, classes, methods, macros, constants, and more are listed in the Symbol Window. There is a small icon to the left of each item in the Symbol Window list, which describes the type of the symbol.

Source Insight scans your file in the background and dynamically updates the Symbol Window. If you type in a new declaration, the symbol will appear right away in the Symbol Window.

Symbol Windows appear on the side of each source window to allow easy navigation within each file, and to provide a quick overview of the file. You can also drag symbols from one Symbol Window to another.

The Symbol Window also displays #ifdef-#endif nesting levels and symbol type icons for quick identification and programmer orientation.



At the bottom of the Symbol Window is a small toolbar. There are controls for sorting the list and a button for running the Browse Local File Symbols command. You can right-click on the Symbol Window to bring up its shortcut menu.

Selecting Symbols Quickly

Click in the Symbol Window to navigate.

To quickly jump to a symbol in a file, just click on the symbol name in the Symbol Window list. To select the whole symbol declaration in the source file window, double-click on the symbol name in the Symbol Window. You can also double-click in the selection bar at the left edge of the source file window to do the same thing.

When the source file window is active and you are typing or selecting text, the current enclosing function, method, or symbol is highlighted in the Symbol Window.

Typing into the Symbol Window

Type into the box at the top of the Symbol Window to filter the list.

You can activate the Symbol Window with the keyboard by running the Activate Symbol Window command (Alt+L). This moves the cursor to the text box at the top of the Symbol Window. When you type in that text box, the list is filtered based on what you type. For example, if you start typing, "Insert", then only items that begin with "Insert" will appear in the list. Press Enter to jump to the selected symbol, or Esc to cancel.

Rearranging Code in the Symbol Window

To drag functions around, click in the Symbol Window icons and drag them.

The Symbol Windows gives a quick overview of what is in the file. It also allows sophisticated editing. If you click on the icon to the left of a symbol name in the list, then you can drag that symbol around. You can drop it at another location in the list to move it, or you can drag it into any other file window, or the Clip Window.

You can also drop text onto another Symbol Window. The text will be pasted just before the symbol that you highlighted in the target Symbol Window.

The right-click shortcut menu contains symbolic cut, copy, paste, and duplicate commands. This makes it easy to do symbolic editing operations.

Customizing the Symbol Window

The Symbol Window can be sorted by name, line number, and symbol type by clicking the sorting icons on the Symbol Window toolbar, or by running the Sort Symbol Window commands.

You can control what types of symbols appear in the list by running the Browse Local File Symbols command and selecting the desired Criteria check boxes. For example, if you don't want to see Enums in the list, turn off the Enums check box.

You can resize the width of the symbol window by clicking on the right edge of it and dragging. Once you set up the size of a Symbol Window, right-click on it and select "Record New Default Properties" to save the current setting as the new default. All new Symbol Window created from this point on will use the new width.

The right-click shortcut menu of the Symbol Window contains more commands, including Record New Default Properties. This command takes the current sorting, list filtering, and width, and makes that be the new default for all new Symbol Windows that are created.

Showing the Symbol Window Per Document Type

Document types determine whether the Symbol Window is used.

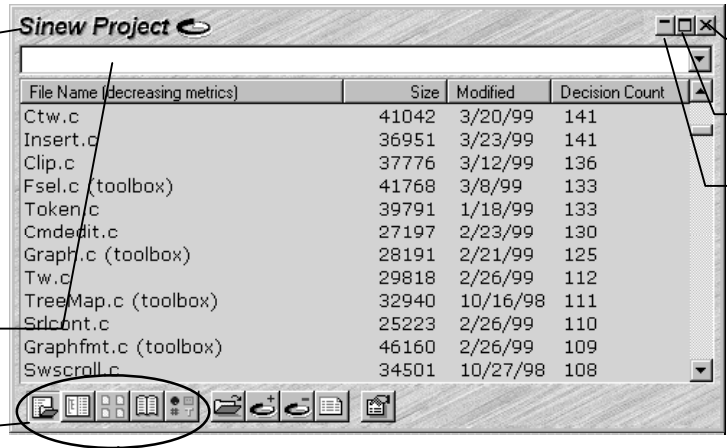
You can control whether a symbol window is used or not for a particular document type by selecting Document Options. Each document type has a "Use symbol window" check box.

Symbol Window Properties

The Symbol Window Properties command, which appears on the Symbol Window toolbar, lets you customize all the Symbol Windows. You can change the color, font, and type of symbols displayed in the list.

Project Window

The Project Window appears when you run the Open command or run the Project Window command (View menu). The Project Window lists all the files and symbols in the project, and it allows you to open files quickly; regardless of what directory they are in.



The project name

You can type a file name in this edit field. As you type, the list is filtered down to show only files that match what you type, regardless of directory.

Tool Bar

Close button

Maximize button

Roll-Up button

These buttons switch the Project Window view: The views are:

1. File List View – lists all files in the project.
2. File Directory View – lists files by directory.
3. File Type View – lists files by document type.
4. Symbol List View – lists all symbols In the project.
5. Symbol Class View – lists symbols by class and type.

The Project Window can be either docked to a side of the Source Insight main application window, or it can float in front.

At the bottom of the Project Window is a small toolbar, with buttons for opening a project, and adding and removing files from the project.

Opening Files Quickly

To open a file, double click on the file name in the Project Window. Right-click on the Project Window to bring up the Project Window shortcut menu. Typing into the text box at the top of the Project Window filters the list down to only files that match what you type. The directory where the file exists is not important most of the time, and it is not used to match the file with what you type. You can type just the leaf name of a file without knowing what directory it is in. Most files in your project can be open with just a few keystrokes.

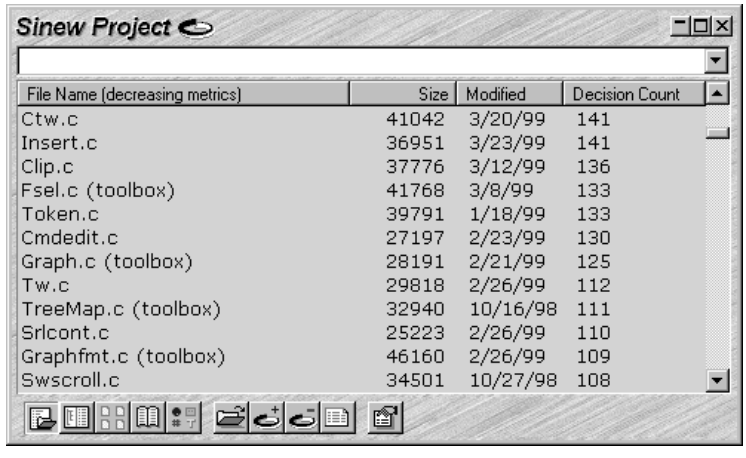
Project Window Views

The Project Window has five different views:

1. File List View – lists all files in the project.
2. File Directory View – lists files by directory.
3. File Type View – lists files by document type.
4. Symbol List View – lists all symbols in the project.
5. Symbol Class View – lists symbols by class and type.

File List View

File List View shows all files in the current project, in a flattened list. You can also type wildcards and change working directories directly by typing into the text box.


Sinew Project
File Name (decreasing metrics) Size Modified Decision Count
Ctw.c 41042 3/20/99 141
Insert.c 36951 3/23/99 141
Clip.c 37776 3/12/99 136
Fsel.c (toolbox) 41768 3/8/99 133
Token.c 39791 1/18/99 133
Cmdedit.c 27197 2/23/99 130
Graph.c (toolbox) 28191 2/21/99 125
Tw.c 29818 2/26/99 112
TreeMap.c (toolbox) 32940 10/16/98 111
Srlcont.c 25223 2/26/99 110
Graphfmt.c (toolbox) 46160 2/26/99 109
Swscroll.c 34501 10/27/98 108

Using syllable matching, you can type part of a file name to locate a file, without bothering with the directory name.

To filter the list with a wildcard, type the wildcard and press Enter.

If you type a wildcard specification and press Enter, then the file list will be filtered down to match that specification. For example, if you type *.c and press Enter, you will see all *.c files in your project, regardless of directory. To remove the wildcard, press * (asterisk) and press Enter.

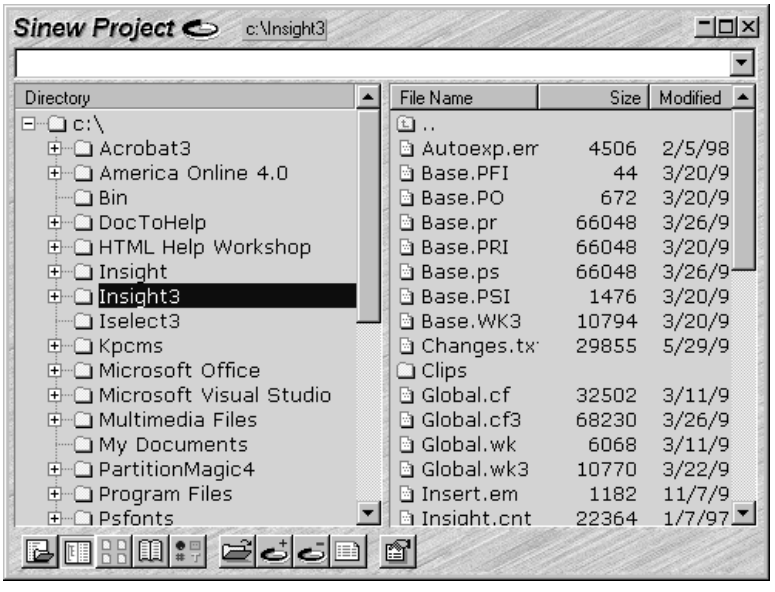
Browsing Non-Project Files

If you want to browse your disk and see files that are not necessarily part of your project, type dot (.) and press Enter. The current working directory contents will fill the list. To return to the "project-only" view of the files, type ** (two asterisks) and press Enter. Alternatively, you can switch to the File Directory View of the Project Window.

File Directory View

File Directory View shows the directory structure of your disk.

File Directory View shows directories and files. This allows you to perform some basic directory and file maintenance, and to open non-project files easily.



Symbol List View

Symbol List View shows all the symbols in the project.

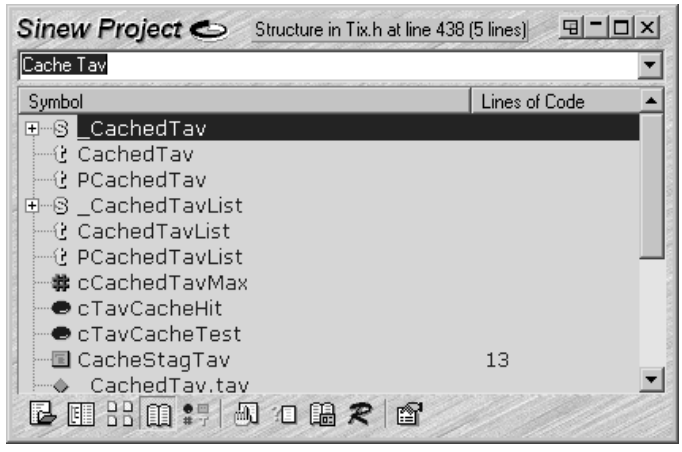
Symbol List View shows a list of all symbols in the project symbol database. This is similar to what used to be shown in the Browse All Symbols dialog box.

To locate a symbol quickly, type a part of the symbol name and the list will be filtered down as you type. For more information, see "Symbol Syllable Indexing" on page 103.

You can also perform a regular expression search for symbol name by prefixing the regular expression with a question mark (?). For example,

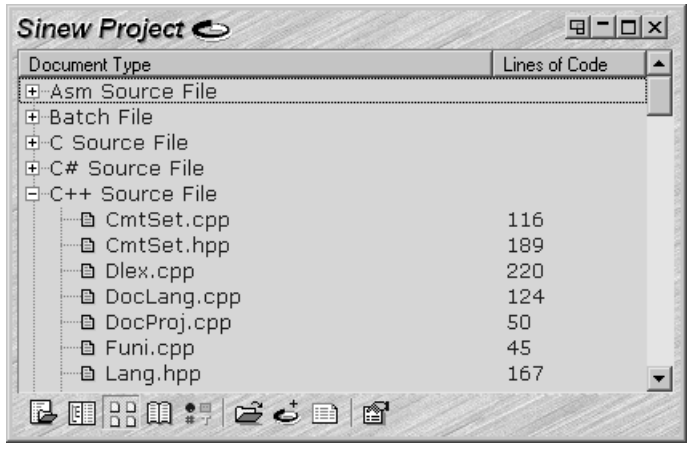
```
?Insert.*Stack
```

will find all symbols that have "Insert", followed by zero or more characters, followed by "Stack".



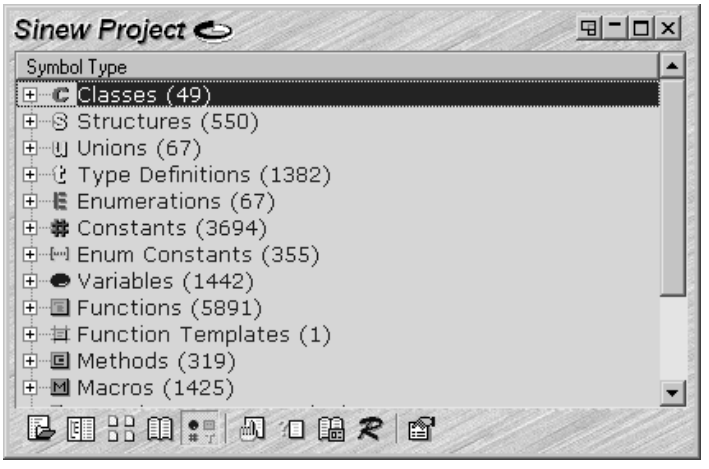
File Types View

File Type View shows a breakdown of files by document type.



Symbol Class View

Symbol Class View shows a breakdown of symbols by symbol class and type.

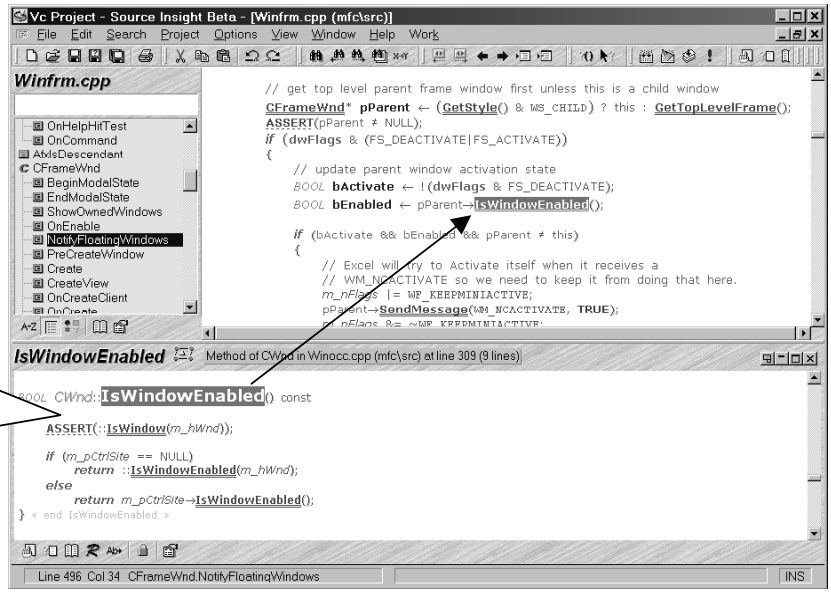


Context Window

The Context Window is a Source Insight innovation that automatically provides relevant information while you are viewing and editing your source code.

The Context Window shows symbol information, based on context.

The Context Window is a floating, dockable window that displays contextual information while you type or click on things. For example, if you click on a function call, the Context Window will display the function's definition. If you start typing, the Context Window will display a filtered list of symbols in your project that match what you've typed.



The Context Window renders information about the selected identifier in the source file window.

You can toggle the Context Window on and off by running the Context Window command. The Activate Global Symbol List command makes it visible and then sets the focus on the Context Window text box so you can type the name of a symbol to locate it in a list of symbols, similar to the Browse Project Symbols dialog box.

Previewing Files

The Context Window shows file previews when selecting files in the Project Window.

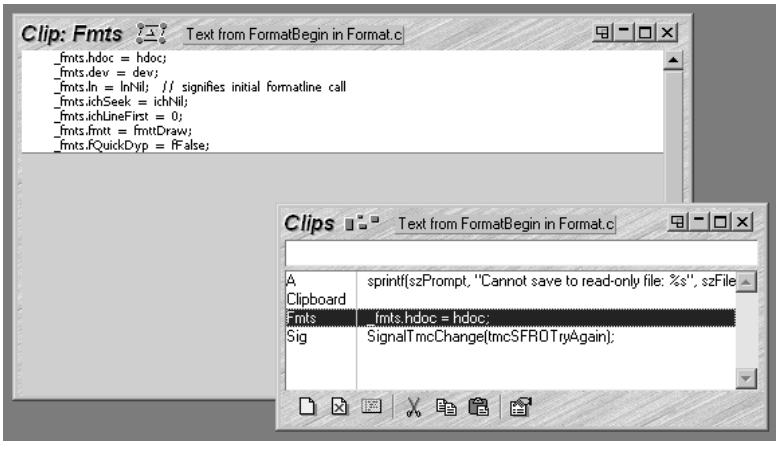
If the Project Window is in front, the Context Window shows the file currently selected in the Project Window. If the Clip Window is in front, then the Context Window shows the clip contents for the selected clip.

Showing Declarations

When you click on a symbol in a source file window, the Context Window will show you the symbol's declaration automatically. If you start typing, the Context Window will show you all the symbols defined in your project that match the first characters you've typed. Functions and other symbols show up in the Context Window along with their parameters and other definition information.

The Context Window determines what type of symbol you are clicking or typing. For example, if you click on a variable, it will show you the declaration of the variable. If the variable is a data structure instance or pointer, then the Context Window will show you the structure.

The Context Window also tracks selections in other types of windows, such as the Project Window, Relation Window, and Clip Window.



The Context Window (background) is displaying the contents of the selected clip from the Clip Window.

To summarize, the Context Window tracks the following:

Your Action	Context Window Result
Selecting (clicking) on an identifier in a source file window	Shows symbol definition based on context. If more than one symbol matches, the Context Window will show a list of matches. If only one symbol in your project matches, then the Context Window will show the declaration of the symbol.
Typing in a source file window	If the auto-completion window is not enabled, then the Context Window shows prefix matches, or symbol definition if unique. For example, if you typed "Insert", then the Context Window will show all symbols that begin with "Insert". It there is only one symbol named "Insert", then the Context Window will show the declaration of "Insert".
Selecting a file in the Project Window	Shows the file contents
Selecting a clip in the Clip Window	Shows the clip contents
Selecting an item in the Relation Window	Shows the selected symbol's definition. If the symbol in the Relation Window is a reference, then the Context Window shows the location of the reference.

Decoding Base Types to Show Structures

The Context Window dynamically decodes base types.

The Context Window also can determine the "base" structural type of a symbol by climbing up the type hierarchy. That is, typedefs are decoded all the way back to the base structure type. For example, if you have code like this:

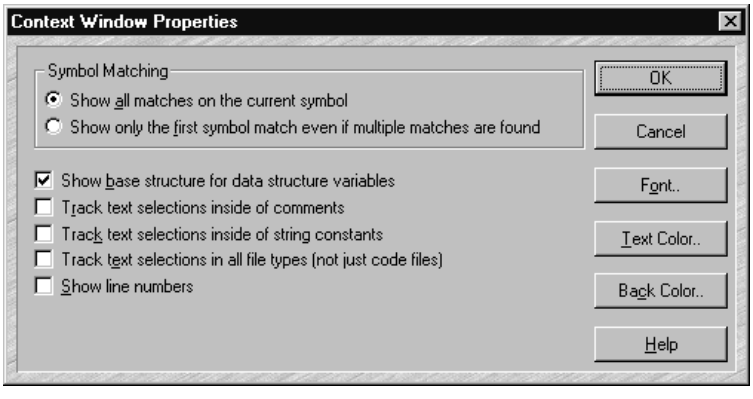
```
struct S { ... };
typedef S T;
typedef T *PT;
PT ptvar;
.
.
ptvar->foo....
```

If you select the `foo` in `ptvar->foo`, then the Context window will find the declaration of `PT ptvar`, and decode the type hierarchy until it finds the `struct S`, and it will display the declaration of the field `foo` within `struct S`.

The Context Window also decodes class hierarchies dynamically. That is, it will travel up the class derivation hierarchy looking for members.

Customizing the Context Window

The Context Window Properties command allows you to change the Context Window settings. You can turn off the base type decoding mentioned above as well as other things. (See Context Window Properties)


Context Window Properties
Symbol Matching
Show all matches on the current symbol
Show only the first symbol match even if multiple matches are found
Show base structure for data structure variables
Track text selections inside of comments
Track text selections inside of string constants
Track text selections in all file types (not just code files)
Show line numbers
OK
Cancel
Font..
Text Color..
Back Color..
Help

Relation Window

You can use the Relation Window to see function call trees, and reference trees.

The Relation Window is a Source Insight innovation that shows the relationship between the currently selected symbol and other things. It works like the Context Window by tracking what you are doing and showing relationship information automatically. The View > Relation Window command toggles the Relation Windows on and off.

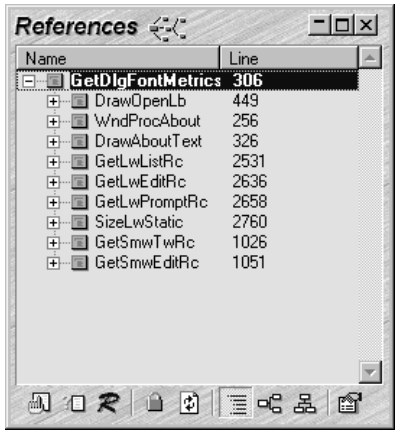
The Relation Window can show function call trees, class hierarchies, aggregate members, and more.

Outline and Graph Views

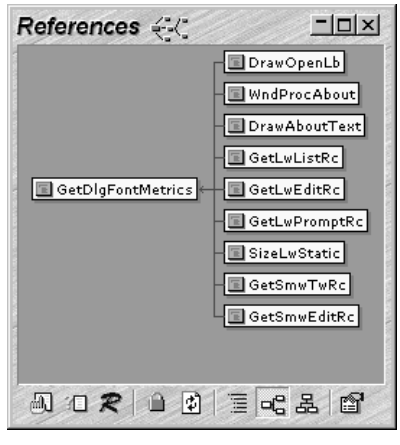
The Relation Window can display in Outline or Graph views.

The Relation Window has two types of views: Outline view and Graph view. The Graph view shows symbols as graph nodes with lines connecting them. The Relation Graph Properties command (available on the Relation Window right-click shortcut menu) gives you control over the appearance of the graph view.

Outline View



Graph View



Relationship Types

The Relation Window can show different relationships.

The relationships fall into three general categories, listed from computationally the fastest to slowest:

- **Contains** – show the contents of the current symbol. For example, the members of a struct.
- **Uses** – show what other symbols are referred to by the current symbol. For example, functions that are called by the current function.
- **References** – show what other symbols refer to the current symbol. For example, functions that call the current function.

Relation Window Performance

The Relation Window requires some processing. Some relationships are slower to compute. For very large projects, the "References" relationship will be the slowest to compute.

Relationship Rules

The Relation Window uses rules to decide what information it shows.

The relationship shown depends on the type of symbol. You can specify what relationship is shown for different symbol types in the Relation Window Properties dialog box. For example, you could set the relationship viewed for functions to "Calls", and the relationship viewed for classes to "Inheritance".

Each time the Relation Window expands a symbol to show a new level, the relationship represented by the expansion is based on the type of symbol being expanded. That means each Relation Window can potentially show multiple relationships.

Call Graphs

The function call graph relationships can be filtered to show only what you consider the most interesting paths. The "Call Graph Filtering" button in the Relation Window Properties dialog box takes you to a dialog box that lets you control the filtering by specifying particular functions you want to exclude. You can also filter functions out by code metrics constraints.

Note that Source Insight considers C macros legitimate function-like symbols, and so C macros may show up in a call graph. You can filter them out in the Call Graph Filtering dialog box if you want.

Multiple Relation Windows

You can use more than one Relation Window to show different relations at the same time.

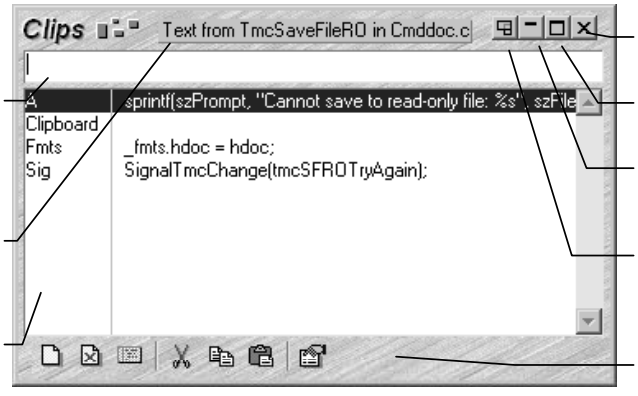
You can have more than one Relation Window by right-clicking on a Relation Window and selecting the New Relation Window command. Having two or more Relation Windows lets you view multiple types of relationships, or track different targets at the same time. For example, one window could show you what functions are called by the selected function, and another window could show you what functions make calls to the selected function.

Relation Window Properties

The Relation Window Properties command is accessed from the Relation Window toolbar or shortcut menu. You control what relationships are shown from this command, and how the window displayed. See "Relation Window Properties" on page 305.

Clip Window

The Clip Window is a floating and dockable window that displays clips. You can drag and drop text onto the Clip Window and drag text from the Clip Window onto your files.


Clips
Text from TmcSaveFileRO in Cmddoc.c
A
Clipboard
Fmts
Sig
sprintf(szPrompt, "Cannot save to read-only file: %s", szFile
_fmts.hdoc = hdoc;
SignalTmcChange(tmcSFROTryAgain);
Type into this field to select a clip. This list is filtered as you type.
Other clip information can appear at the top, telling you where the clip came from.
List of all open clips. Each item shows the first line of text in the clip
Close button
Maximize button
Roll-Up button
Transparency button (Win2K only)
Tool Bar

What Is A Clip?

Clips are useful for rearranging code, and for boilerplate text.

Clips are clipboard-like documents. In fact, the Clipboard is considered a clip in Source Insight. Instead of having only one clipboard, Source Insight lets you have many clips. A clip is like any other file. You can edit it the same as any other file. The difference is that clips are automatically saved between sessions and clips can be pasted easily with the Paste From Clip command.

Clips are useful for rearranging code, especially between many files. Clips are also useful for boilerplate text that you often want to insert.

You can toggle the Clip Window on and off by running the Clip Window command, or by running the Activate Clip Window command, which makes it visible and then sets the focus on the Clip Window text box.

Creating a New Clip

To create a new clip, run the New Clip command (located on the File menu, and on the Clip Window toolbar). A new source file window will open. You can type and edit this file like any other. When you close it, the clip will be retained in the Clip Window until you delete it.

To create a new clip, click and drag text onto the Clip Window.

You can also create a clip anytime by using the Copy to Clip command, or by dragging and dropping text onto the Clip window. You can also drag a file from Explorer onto the Clip Window to open a file as a clip.

Clip Storage

Clips are automatically saved to the Clips subdirectory of your Source Insight program directory. Any text file that you place in that Clips subdirectory will be automatically loaded when Source Insight starts up.

Search Results Window

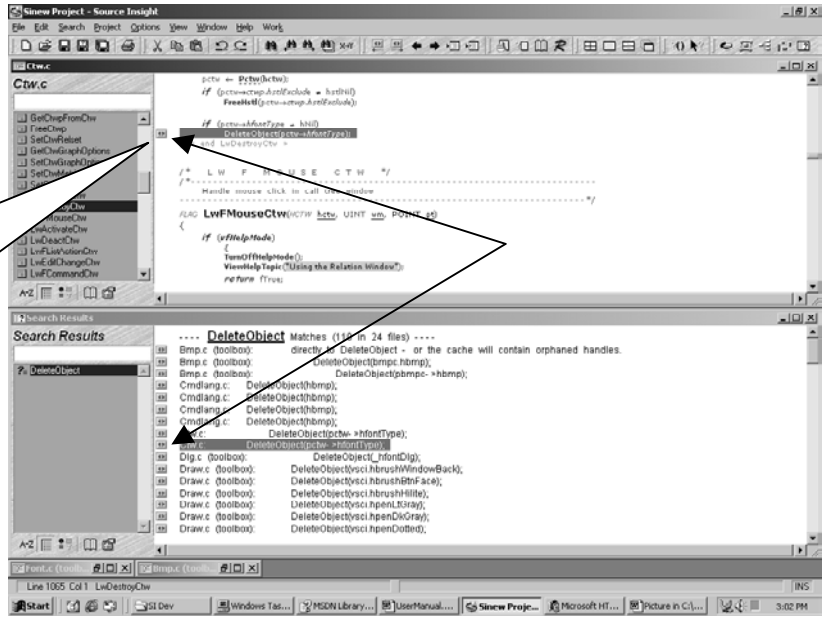
Multi-file search and replace operations output to the Search Results window.

The Search Results window is created whenever you run the Search Files or Lookup References commands. Each line in the Search Results window corresponds to a match in some file at some line number. Matches listed in the Search Results window contain source links, which are hypertext-like links to the locations where the matches were found.

Source Links

In the screen shot below, you can see the Search Results window contains source link icons on each match line. Each source link corresponds to a location in a file at some line number. The window above the Search Results is the target location of one of the source links.

Source Links connect two locations. These links connect Search Results with matches.



The Search Results window has a source link on each listed match.

Each set of matches in the Search Results window has a search heading that lists the pattern that was used to search, followed by a match count. The search headings are parsed by Source Insight and each search pattern appears in the symbol window on the left side of the Search Results window to provide an overview.

For example, a typical search heading might look like:

```
---- GetStockObject (23) -----
```

meaning that 23 occurrences where found.

The Search Results window is actually just another file buffer that you can edit. You may freely delete lines from the window to trim down the results.

Source Insight Concepts

This chapter is a guide to Source Insight's features and concepts. You will get much more out of Source Insight if you take a little time to scan this chapter.

As you read this chapter, you will become familiar with Source Insight's features. Later, as you explore Source Insight's commands, you can refer to the Command Reference chapter for information on specific commands.

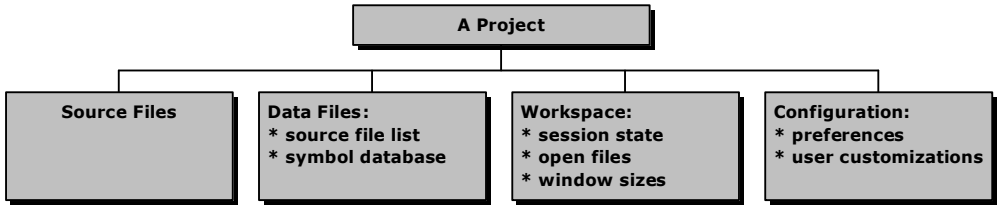
Projects

A project is a collection of source files.

Source Insight is built around projects. A project is a collection of source files. After you create a project, you add files to it. Source Insight records what files are in the project by keeping a simple file database for the project.

As you create new files, they can be added to your project when you save them. If new files appear in your project directory or subdirectories, they can also be added automatically to your project by running the Synchronize Files command, or by letting Source Insight synchronize automatically in the background.

When a project is open, some of Source Insight's operations change or are enhanced. For example, the Project Window lists all files in a project, regardless of directory.



A project automatically contains a symbol database, which is maintained by Source Insight. Except for adding source files to your project, you do not have to generate any other "tag" files. Source Insight does that automatically.

Each project has its own workspace. The workspace contains session information, such as the list of files that are open and window positions.

Each project can have its own configuration, or it can use the single global configuration. The configuration contains your customizations, which includes many of the options set via the Options menu.

Throughout this documentation, all discussions assume that you have a project open, unless otherwise stated. Whenever there is a difference in the way a command works with and without a project open, it will be noted.

The Current Project

The current project is the single project that is open in an instance of Source Insight.

The project that is open, if any, is referred to as the *current project*. You may only have one project open at a time, however you can have different instances of Source Insight running; each of which can have a different project open.

Sometimes, Source Insight will open other secondary projects to search for symbol declarations. However, each instance of Source Insight only allows you to open one project at a time.

How Source Insight Uses Projects

Source Insight projects have several important advantages:

- A project logically groups related files.
- When you specify a file to be opened, you don't have to tell Source Insight the file's drive or directory. Source Insight "knows" where all the project files are, even if they are in different directories or drives. (See Also: The Command Line).
- Source Insight maintains a symbol database, which contains data about all symbols declarations in the project. You can use Source Insight to locate symbols very quickly. When source files are saved, the symbol database is automatically updated incrementally so that Source Insight always "knows" where a symbol is. When files are changed external, by a source control system for example, Source Insight will automatically synchronize those files with the project symbol database.
- Source Insight can show symbol relationships in the project, such as call trees, reference trees, and class hierarchies.
- Source Insight maintains a reference index, which greatly speeds up project-wide searches for symbol references. The reference index is updated incrementally as you edit and save your files.
- Each project has its own workspace. When a project is opened, all the session state is restored. When a project is closed, all open files are closed and the workspace is saved.
- Each project can have its own configuration file. This means that each project can have its own set of menus, keyboard assignments, and screen colors.

Creating a Project

Use the Project > New Project command to create a new project. You must give the project a name and specify where the project data directory is.

Project Directories

The *project data directory* is the directory where you told Source Insight to create the project. The Source Insight project data files are stored there. That includes the .PR file, which is the main project file. It does not have to reside in the same directory as your source files.

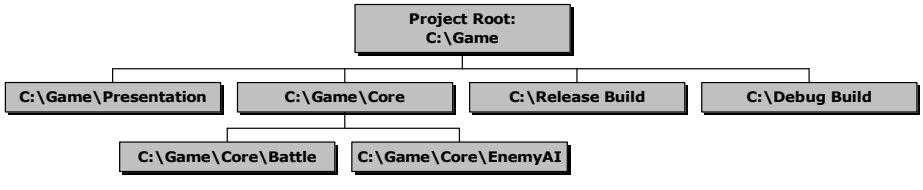
The *root directory* is what you consider the "home" directory of your source files.

The *project root directory* is what you consider the "home" directory of the project. The project's root directory is typically the topmost, or "root" directory that contains most of the source files. Source Insight normalizes project file names relative to this directory. (See figure below). By default, Source Insight makes the project root directory the same as the project data directory. You can actually set the root directory to any location on your disk, after the project is created.

See "Where Should You Create A Project?" on page 280 for more information about choosing a location for your project.

Once a project is created in a given directory, you can add files to it from any directory and any drive, including network drives, and UNC paths.

As an example, let's say we are creating a project for a game program. We want to divide the source files into categories and create a directory for each category. We create the necessary directories and create a project whose root directory is C:\Game.



The project root directory in our game example is C:\Game. We have source code in the "Presentation", "Core", and "Core" subdirectories. Our Source Insight project will include files from all these directories.

Normalized File Names

When Source Insight displays a file name and the file is part of a project, it arranges the name and path to make it easier to see and select the base file name without all the directory paths getting in the way. This process is called *normalizing the file name*. This is an important feature because many projects have files spread out across multiple subdirectories; and "flattening" out the directory tree makes it easy to type and select the most significant part of file names.

A normalized file name always begins with the leaf file name, followed by the directory path in parentheses.

A normalized file name always begins with the leaf file name, and it's followed by the directory path in parentheses. Furthermore, the directory path shown is relative to the project's root directory, unless the file is on a different drive. If the file is on a different drive, or not part of the project root directory tree, then the full path is displayed in parentheses.

Here are some examples using the game project discussed above:

If the file path is:	The file name is displayed as:
C:\Game\File.c	File.c
C:\Game\Core\File.c	File.c (Core)
C:\Game\Core\EnemyAI\File.c	File.c (Core\EnemyAI)
C:\SomeDir\File.c	File.c (C:\SomeDir)
D:\OtherDir\File.c	File.c (D:\OtherDir)

The Project List

The Project List contains a list of the recently created projects on your machine.

As you create projects, Source Insight keeps track of them in the Project List. There is only one Project List and it is created the very first time you run Source Insight. The name of the file is Projects.db3, and it is created in the Source Insight program directory. The Project List stores the names of all the projects created or opened on your computer, including the directory where they were created.

Adding Files to a Project

Once having created a project and it is open, you next need to add source files to the project. This can happen two ways.

If you created a new file in Source Insight and save it for the first time, Source Insight will ask you if you want to add the file to the current project. This will be the most natural way to add a file if you are writing new code and are creating new source files a lot.

Use the Add and Remove Project Files command to add your files to the project.

If you already have existing source files and you want to add them to the current project, use the Add and Remove Project Files command. This command allows you to add any existing files, including whole directory trees, from anywhere on your disk to the current project. See "Add and Remove Project Files" on page 174, and "Add File List" on page 179.

Adding a file to a project has the following effects:

- The file name is stored in the file name database for the project. Whenever Source Insight displays a list of files, that file name will be in the list. Therefore, for example, when you use the Open command, the file name will be in the list box.
- The file is parsed for based on its language type. All the symbol definitions found are added to the project's symbol database. The language parser used for each file is determined by its document type. See also Document Options.
- The file's modification date is recorded in the file name database, so that Source Insight will know to synchronize the project symbol database if the file was modified outside of Source Insight; for example by a source control system.
- The way the name of the file is displayed is changed. The file name becomes "normalized" to the project's root directory.
- The file will become part of the project code base, which is searched when showing symbol relations, such as call trees.

Removing Files from a Project

To remove a file from the current project's file list, use the Add and Remove Project Files command. When a file is removed from a project, all the symbols found in that file are removed from the

project's symbol database. Source Insight will *not* actually delete the file from the disk.

Closing Projects

To close the current project, use the Close Project command. Closing a project is a lot like quitting; Source Insight asks you if you want to save each file you have opened that you've changed, then it closes all files. Instead of actually quitting, Source Insight continues with no project open.

Opening Projects

To open a project, use the Open Project command.

To open a different project, use the Project > Open Project command. Source Insight only allows you to have one project open at a time, so if you already have a project open, it will ask you if you're sure you want to close the current project. Assuming you do close the current project, the Open Project command will display a list box of existing projects from which to choose.

When a project is opened, the project's configuration file and workspace file are loaded, which means the display, menu, and keystroke configuration may change and the files you had open in the previous session with the project are reopened.

When a project is opened, the current working directory is changed to the project's root directory.

See "Open Project" on page 282.

Removing a Project

To remove a project, use the Remove Project command. This command removes all the project files that Source Insight creates and associates with the project, and it asks you if you want to delete the source files that are in the project. See "Remove Project" on page 311.

Project Settings

Set project indexing options with the Project Settings command.

The Project Settings command allows you to set various options that govern the current project. If no project is currently open, then the Project Settings command allows you to set the default options inherited by subsequently created projects.

When you create a new project, the Project Settings dialog box appears.

You may specify whether the project has its own private configuration, or if it uses the global configuration file. You can also indicate where the project's root directory is, and what types of symbol information should be indexed.

See "Project Settings" on page 294.

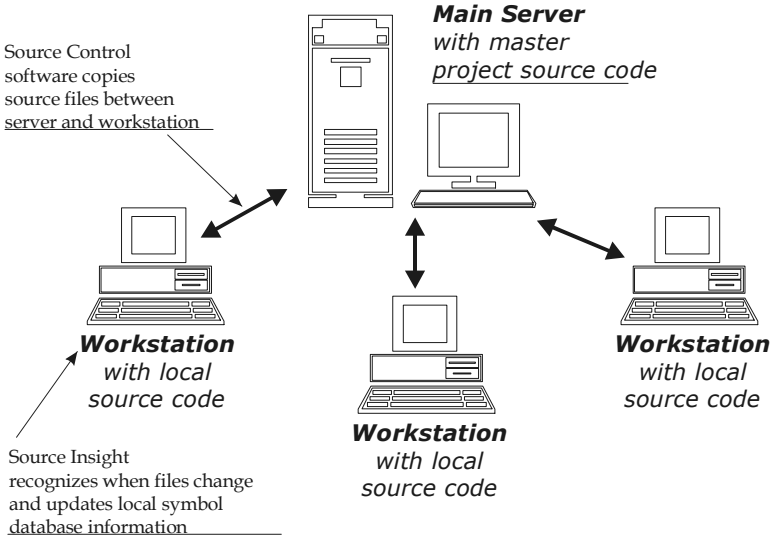
Working in a Team

Source Insight is designed to work well in team programming situations. As team programmers contribute to the code base, Source Insight automatically recognizes their contributions and updates its symbolic information. Moreover, as large amounts of new code are added to a project, or moved from module to module, you will appreciate Source Insight's ability to keep track of everything for you.

Networking

Adding local files to your project is recommended.

Source Insight works best if each team member has their own local copies of the project source files on their own local machine. Typically, the "master" copies of the source code are kept on a central server.



Adding Remote Files to a Project

For whatever reason, you may want to access your source files directly from the project server, not from local copies of the file. Of course, you are free to open any file on the network directly. However, keep in mind that you may be locking other people out of the file by having it open, or otherwise causing contention over the file. In addition, you wont get the benefit of Source Insight's project features unless the remote file is added to your project.

If needed, you can add source files to your project from remote drives.

One way to have a project that refers directly to files on the server is to create a project locally on your workstation and add the files from the remote server to your project using the Add and Remove Project Files command. This way, the Source Insight symbol database files are stored locally on your machine, but the source files are still just on the server.

Using the Project Settings command, you should specify the remote source code directory, on the server, as the project root directory. That way, files will be displayed relative to the main source directory, not relative to your local project data file directory.

Source Control

In a typical networking environment, each developer has his or her own local source files on their workstation machine. When a developer wants to make a change to a file, they first "check it out" using source control software. After the file is checked out, they edit a local version of the file with Source Insight. When the developer is satisfied that they want to keep the changes made, they check it back in using their source control software. This effectively copies the file from the developer's local machine back to the main project server.

Now, when another developer wants to get the "latest and greatest" version of the source files, they use the source control software to "synchronize" their local directories with the main project server. This effectively copies newer files from the project server to the developer's local workstation machine.

Source Insight recognizes when files change by checking each file's "last-write" timestamp, and the file's size. When it detects that a file has changed, it re-parses the file and updates the Source Insight symbol database on the developer's local machine.

Source Control Commands

There are standard custom commands for source control.

Source Insight has several standard Custom Commands built-in to handle source control operations. The Options > Custom Commands dialog box lets you edit the commands. By default, they are setup to support Microsoft® Visual Source Safe™. However, you can easily change them to support other source control, or version control systems.

The source control commands appear in the table below. Their exact meanings are really based on how you choose to set them up in the Custom Commands dialog box. Source Insight defines the

following command semantics.

Command Name	Action
Check Out	Checks the current file out of the Source Control project, so that you can edit it.
Check In	Checks the current file into the Source Control project. You should use Check In after you are finished editing a file, and want to put it back into the main Source Control project for other team members to access.
Undo Check Out	Reverses the action of a Check Out. This does not check the file back in.
Sync to Source Control Project	Updates all the files in your local project so they are current with respect to the Source Control project.
Sync File to Source Control Project	Updates the current file so it is current with respect to the Source Control project.

The Source Control toolbar contains buttons for each of the source control commands.

Symbols and Projects

Source Insight parses symbol definitions dynamically out of your source files while you edit. Symbol information is stored on disk in an indexed symbol database, which is integral to the project.

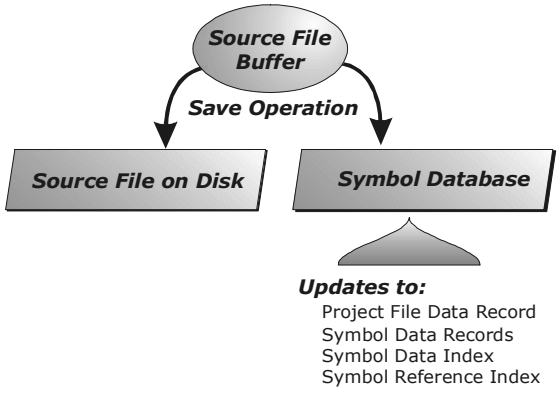
The symbol database is automatically updated when files change.

The symbol database is updated incrementally as you open and save files. Files that are changed by other project team members are also automatically synchronized with the symbol database in the background.

The language parser used for each file is determined by its document type. (See Also: "Document Types" on page 100.) Source Insight's language parsers recognize a wide variety of declarations.

Source Insight uses sophisticated, error-tolerant pattern matching parsers to find symbol declarations in your source files. This goes far beyond what other batch tools, such as "ctags", can do. You also can specify custom regular expression patterns to use when parsing symbols from your files by using the Preferences: Languages command and clicking the Properties button.

When a file is added to a project, or a file is saved, Source Insight determines what symbols are defined in the file, and incrementally updates the project symbol database stored on disk.



File Names Are Like Symbols

Source Insight also treats file names as symbols. Thus, file names may be specified wherever a symbol may be. For example, you can type a file name (with extension) in any of the Browse… dialog boxes to open a file. You can also click on a file name in a #include statement and use the Jump To Definition command to open the file.

Synchronizing Project Files

Synchronizing means the symbol database is updated for all modified files.

Sometimes files are edited without using Source Insight. For example, you may be using a source control system that updates files on your machine, or you may have files that are machine generated by your build process. When that happens, Source Insight has to re-parse those out-of-date files to bring the Source Insight project up to date. That process is called *synchronizing the project*. Normally, this is done automatically in the background for you.

The synchronize process ensures that the entire project is up to date. It scans each file in the project and updates the symbol database for each file that has been modified since Source Insight had the file open last. Files that were part of the project that don't exist anymore are removed from the project. As an option, you can have the synchronization automatically add new files that it finds to the project.

You can synchronize a project in one of two ways:

- You can turn on the "background synchronization" option in the Preferences: General dialog box. Synchronizing the project will happen in the background while you continue to edit in Source Insight. This option is on by default.
- You can run the Synchronize Files command, which synchronizes the project on demand.

Normally, if you open a file that has been modified since Source Insight had it open, the symbol database is updated automatically when you open the file. By automatically synchronizing, the update is transparent to you. However, if a symbol is moved from one file to another, you may find that Source Insight loses track of where the symbol is, unless *both* files are in sync with the project. In addition, if new symbols have been defined, Source Insight won't know about them until the containing file is synchronized.

The Project Symbol Path

Source Insight will search all the projects on the *project symbol path* if it doesn't find a declaration in the current project.

You might have a set of headers files from a library that you often use with multiple projects. You could add these files to each of your projects, but that would be redundant. A better solution is to create a single project for each set of common header files. These projects can be put onto a "path" and Source Insight will search the path whenever it looks up a symbol and can't find it in your project. This path is called the *project symbol path*.

When Source Insight looks up a symbol, it searches the currently open files, the project symbol database, and the projects in the project symbol path.

The project symbol path is a delimited list of projects that Source Insight will search through when looking up a symbol. The project symbol path is located in the Preferences: Symbol Lookups dialog box. The project symbol path enables you to create smaller, self-contained projects, but still have the ability to locate symbols in other projects. The Base project (described below) is the final place Source Insight looks to find symbols; it is implicitly at the end of the project symbol path.

The project symbol path is only used to locate symbol definitions. It is not used to locate symbol references, or used by Search Files, or Smart Rename. Those operations only work on the files in the current project.

Searching the Symbol Path

The project symbol path is searched when searching for a symbol's definition.

Source Insight searches all open files and the symbol database for the current project. If the symbol is not found, then *every* project in the project symbol path is searched. If the symbol is still not found, then the Base project is searched.

If more than one symbol is found with the specified name, then Source Insight will ask you to pick from a list of the matching symbols, their locations, and types.

You can enable the "Always search symbol path" option in the Preferences: Symbol Lookups dialog box. When this option is on, all projects in the symbol path are searched *every time* Source Insight looks up a symbol, even if the symbol was already found in an open file or the current project. You may want to turn this option on if you want to see if there are redundant (or at least like-named) symbols in your project and any others on the project symbol path.

When No Project is Open

When no project is open, Source Insight searches the files that are currently opened. If the symbol is not found, then *every* project in the project symbol path searched. If the symbol is still not found, then the Base project is searched.

The Base Project

The Base project is searched as a last resort.

The first time you run Source Insight, it automatically creates a project named Base. The Base project is the last place that Source Insight will search for symbols. It is implicitly at the end of the project symbol path. In other words, if Source Insight is looking up a symbol, and it can't find it in either an open file, the current project, or any of the projects in the project symbol path, then it looks in the Base project for the symbol.

Note: You do not have to add the Base project to the project symbol path. Source Insight automatically searches it as though it were in the list.

This gives you a convenient place to save common symbols. Any symbol stored in the Base project is visible from any other project. For example, you could add all of the standard C/C++ include files to the Base project. The Base project is also a good place to add your favorite Source Insight editor macro files.

Language Support

Source Insight uses a *language* abstraction to encapsulate the properties of various programming languages. The Document Options dialog box is used to associate a *document type* with a *language*.

Source Insight supports languages by displaying source code with syntax formatting, parsing symbol definitions out of the code, and storing symbolic information in the project's symbol database.

Symbol declaration parsing is controlled by the language type associated with the file's document type.

To see a list of the currently supported languages, use the Preferences: Languages dialog box. New languages are added to the list from time to time in program updates.

Languages in Source Insight are divided into two categories: *Built-In*, and *Custom*.

Built-In Languages

Source Insight contains built-in optimized support for several languages, including C/C++, Java, IDL/ODL, Perl, C#, ASP, Visual Basic, and others.

Most built-in languages support extra features, like finding references and generating call-trees.

Custom Languages

You can also add your own custom language support to Source Insight using the Preferences: Languages dialog box. A custom language is a simple generic language that specifies syntax rules, syntax formatting keywords, and simple parsing expressions. See "Language Options" on page 254.

Custom languages can be exported and imported using the Preferences: Languages dialog box.

How to Add a Custom Language

Adding support for a new language is basically a two-step process:

1. Add the language, using Options > Preferences: Languages.
2. Add the Document Type that refers to the language, by using Options > Document Options.

Note that the Document Options dialog still allows you to add a custom parsing pattern directly into the document type properties, in lieu of adding a new language. This functionality will probably be phased out in a future release. You have more control by adding a new custom language and using your document type to point to the new language.

See also: "Document Types" on page 100, and "Language Options" on page 254.

HTML Parsing

Source Insight has special features for handling the HTML Language. The HTML Language parser scans files to locate interesting structural tags. Those tags show up in the symbol window attached to the left side of each source window. The only symbol types added to the project symbol database are the "TITLE" tags. For example, an HTML file might contain this:

```
<TITLE>Programming Wisely</TITLE>
```

It would result in a symbol added to the symbol database as "TITLE: Programming Wisely". The maximum length of a symbol is 79 characters.

HTML and ASP Compound Languages

HTML and ASP are actually compound languages, which may contain embedded scripts.

The HTML Language also supports embedded scripts. Server and client-side scripting blocks will appear using the syntax formatting appropriate for whatever scripting language is used. Elements defined in the scripts are also displayed in the symbol window, and are saved in the symbol database.

You can specify the default scripting language to use in HTML or ASP in the Options > Preferences: Languages dialog box. Click on the Special button and select the default language where it says "Default script language".

C/C++ Issues

If you are using C/C++ for programming, Source Insight has some special features that you should be aware of.

Inactive Code - ifdef Support

ifdef's are respected when condition values are specified in the Edit Conditions dialog box.

Source Insight's C/C++ and Resource File parsers can recognize inactive blocks of code that are disabled at compile-time with #ifdef, #if, and #elif directives. The Options > Preferences: Language dialog box has a set of "Conditions" buttons that allow you to edit the list of known conditional constants. You can also edit the conditions using the Edit Condition command on the right-click menu of a source window.

By default, Source Insight ignores the conditional directives altogether. It attempts to make sense of all branches in a conditional compilation construct. Often, this works well because declarations in the conditional branches do not interfere with each other.

However, sometimes a tricky declaration may be broken in the middle with an #ifdef. This will often confuse Source Insight. For example:

```
void DoThing(
        int param1,
#ifdef ABC
        int param2)
#else
        int param2, param3)
#endif
```

In addition, you may not be interested in code that is inactive. For these reasons, Source Insight lets you specify condition values.

Blocks of code that are inactive are displayed in the "Inactive Code" style. For example:

Inactive code is formatted with the Inactive Code style.

```
#ifdef NEVER
        TluParentRefFromSrl(hsrl, ist, &tlu);
        tlu.hdoc = hdocNil;
#endif /* NEVER */
        return CmdGotoTlu(&tlu) == cmdOK;
```

Conditional Parsing

Conditional parsing applies only to languages that support conditional compilation in Source Insight: C/C++ and Windows Resource files.

Source Insight maintains two types of condition lists.

- Configuration file based condition list. This list is saved in the current configuration file, which contains your customizations. Normally, there is one global configuration file used for all of your projects. The configuration file is stored in the Source Insight program directory.
- Project-Specific condition list. This list is saved with each project. This allows you to have different conditions defined for each individual project. For example, you could have "RETAIL" defined in one project, and "DEBUG" defined in another.

The two condition lists are combined when Source Insight parses a file. The project-specific conditions take precedence over the "global" configuration based conditions.

Condition Values

Each condition can be in one of four states:

- **Active** - the condition is assumed true.
- **Inactive** - the condition is assumed false.
- **Parse Active, Display Inactive** - the condition is assumed to be true, but code inside the conditional block is formatted with the Inactive Code style. This is useful for conditions like "FUTURE_USE" or "NOT_USED_NOW" where the code inside the block is legitimate code for browsing in, but you want the display to show it grayed out as "inactive".
- **Ignored** - the condition is unknown. Source Insight ignores the conditional directive altogether. This is the default case for any unknown conditional directives encountered. An ignored condition is, by definition, not in the condition set.

When you edit the condition list, Source Insight will ask you if you want to re-parse your whole project. You should make all your changes to the condition list first, and then re-parse your whole project. Until your project is re-parsed, the symbol information stored in Source Insight's symbol database will not reflect the changes you made.

Preprocessor Token Macros

You can define how macros are expanded with Token Macros.

Source Insight contains its own preprocessor, which is used when parsing files. The preprocessor macros are called *Token Macros*. They are token substitutions that occur as Source Insight parses a file. They allow Source Insight to handle any special language keywords not known to Source Insight's parsers, and to handle special C/C++ preprocessor substitutions that would otherwise confuse it.

C Macros are normally not expanded by Source Insight.

Source Insight does not expand C/C++ preprocessor macros when it parses your files. Because of this, certain preprocessor macros and constants can fool Source Insight. Therefore, token macros are used to let Source Insight selectively expand some preprocessor substitutions.

One case in point is the set of standard COM (or ActiveX) macros that are used to declare symbols. For example, the STDMETHOD(methodname) macro is used to declare a COM method function. Source Insight ships with a default C/C++ token macro file (c.tom) that has entries for this macro, and many others.

Token Macro Files

Token macro files have a .tom extension.

The token macros are listed in a file with a .tom extension. The global token macro file resides in the Source Insight program directory. The project-specific token macro file, if any, is stored in project's data directory. The project token macro file is combined with the global file, with the project macros taking precedence.

Token Macro Syntax

A token macro file consists of token macros, one per line. The format of a token macro is:

```
macroname  <no text here means macro is a no-op>
macroname  substituted text here
macroname(parameter list) substituted text with parameter
names
macroname(parameter)   text##parameter // concatenates text
; comments begin with a semicolon
```

Some examples of token macros:

```
MyStructure(sname) struct sname
NoOperation
BuildName(name1, name2)       name1##name2
```

Each built-in language parser has a corresponding token macro file. The name of the token macro file for each language is summarized below:

Language	File Name
C and C++	C.tom – a default copy ships with Source Insight.
HTML	Html.tom
Java	Java.tom
Resource Files	Rc.tom
x86 Assembly Language	X86.tom
Perl	Perl.tom

Editing Token Macros

If you want to change the token macros, simply open the token macro file, make your changes, and save the file. Source Insight will recognize that the token macros have changed for the appropriate language. Open files are automatically re-parsed.

Save a token macro file to get Source Insight to recognize the macros.

When you edit a token macro file, you must save it to disk before Source Insight will re-parse your open files. However, Source Insight will not automatically re-parse your whole project. You should make all your changes to the token macro file first, then use the Rebuild Project command to re-parse your whole project. Until your project is re-parsed, the symbol information stored in Source Insight's symbol database will not reflect the changes you made to your token macros.

Project Specific Token Macros

Each project can have its own token macro file, which merges with the global file.

Each project can have its own set of token macro files. Source Insight does not create them automatically, but you can yourself. A project token macro file is saved in the project's data directory. When Source Insight parses a source file, it combines the project token macros with the global set saved in the Source Insight program directory. The project token macros take precedence over the global ones. By adding project specific token macros, you can tailor the token macro expansion for each project individually.

Parsing Considerations

Having symbols found by error-tolerant pattern matching rather than strict language parsing has important implications:

- The source files can have syntax errors. They don't have to be compiled. That means symbols will be found in source code that doesn't even compile or is in an intermediate state, which is most of the time!
- Preprocessor macros are not expanded before Source Insight parses the code. This may sound bad, but in reality it works quite well. Source Insight parses your program at its most abstract level - at the same level as it was written. Among other things, this also allows call trees to contain function-like macros.
- Symbol declarations are parsed from all of the source code, not just what was active at the time of compilation. For example, C/C++ code inside of #ifdef-#endif clauses are also added to the symbol database, even if the #ifdef branch is not active when you compile. This can be a great help if you are working on a multi-state program, and you need to be aware of all cases, not only what the compiler sees. If you do want to omit inactive code blocks, you can define condition values with the Edit Condition command, or in the Preferences: Languages dialog box.
- The text in comments and constants is also indexed for searching. Source Insight is focused on source level code; not just what the compiler transforms into object code.
- All header files are assumed accessible in all source files. Source Insight does not notice what header files are included in each particular source file. Therefore, all symbols are known at all points. This is technically inaccurate with respect to how a compiler will see your program, but in most cases it works well.
- Some programming styles may cause symbols not to be found by Source Insight. The default parsing that Source Insight uses works very well with most programming styles. In the event that you have some declarations types that Source Insight can't recognize, you can add Token Macros, which expand during a preprocessing phase, or you can add a custom parsing regular expression pattern.

Coding Tips for Good Parsing Results

Some coding styles affect parsing correctness.

Because regular C/C++ preprocessing is not performed by Source Insight's parsers, you might want to keep the following thoughts in mind as you write new code.

- Try not to have #ifdef-#endif blocks break up an individual declaration. If it cannot be avoided, and Source Insight doesn't parse the code correctly, you will need to define the

condition value using Edit Condition, so that Source Insight will disable the inactive block of code causing the confusion.

For example:

```
void MyFunc

#ifdef XYZ
    (int param1, int param2)
#else
    (long param1, long param2)
#endif
{
    …
}
```

- Try not to replace standard language keywords or combinations with #define'd substitutions. If you cannot avoid this, then you will need to define Token Macros to support them.

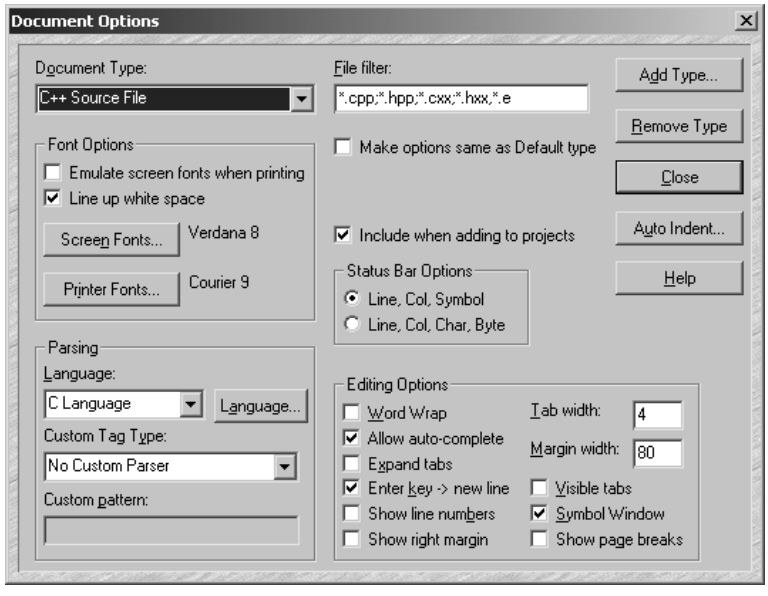
  For example:

```
#define ourpublic public

class D : ourpublic B { … }
```

Document Types

A *document type* is a file classification that is defined with the Document Options command. Source Insight uses each file's name to determine what document type it is. The Document Options command allows you to define new document types or change the built-in types. See Also: "Document Options" on page 219.



Document-Specific Options

The document type is key to determining how Source Insight treats a file. Most importantly, it controls what programming language is associated with each file.

The document type determines the language and editing options for a file type.

A document type also specifies editing and display options, such as the tab width, word wrap, auto-indentation, display font, and others.

Associating Files with Document Types

A file is associated with a document type by its name and/or extension.

The Document Options command associates a document type with a filename wildcard. For any given file, Source Insight determines the document type by matching its name with the filename wildcards specified in all defined document types. For example, *.c files belong to the "C Source File" document type, while *.asm files belong to the "Asm Source File" document type.

Associating Special File Names

The *filealias.txt* file contains file name aliases used to determine the document type.

Some files in your project may not have any file extension. For instance, the standard C++ header files, such as "complex" do not have file extensions. This situation makes it difficult to use wildcard specifications to associate a file with a document type. For that reason, Source Insight uses a special file named *filealias.txt* to create file name aliases for the purpose determining the file's document type.

The *filealias.txt* file is stored in your Source Insight program directory. You can edit this file. By default, it contains the names of the standard C++ header files.

The format of *filealias.txt* is as follows:

```
oldfilename=newfilename
```

This maps oldfilename to newfilename before determining the file's document type. For example:

```
algorithm=algorithm.h
```

When Source Insight sees the file *algorithm*, it uses the alias *algorithm.h* when determining the document type of the file.

Adding Known File Types

Adding new document types makes Source Insight aware of new types of files.

The union of all document types becomes Source Insight's file "vocabulary". That is, when Source Insight shows you the files in a directory, it will only display files that belong to currently defined document types. In addition, when Source Insight automatically adds files to a project, it will only add files that belong to known document types. Therefore, by adding a new document type, you are expanding Source Insight's vocabulary of file types.

Editing the Document Options

The easiest way to edit a file's document options, or to see what document type it belongs to, is to right-click on the file window and select Document Options from the shortcut menu.

Symbol Syllable Indexing

Syllable matching finds partial matches on parts of symbol and file names.

Syllable indexing and matching is a feature that helps you find symbols, even if you are not sure what the symbol's name is. For APIs that maintain consistent naming conventions, you can use syllable matching to find all symbols relevant to a particular topic. By typing a meaningful partial name, you will be able to narrow your search to related items. For example, by just typing "Win", you can see all "Win" related functions.

Syllable matching also works in most type-in boxes that are associated with lists; not just symbol lists.

The symbol database needs to be indexed to find syllables in symbol names.

Source Insight indexes not only the names of each symbol in the database, but it can also index "syllables" within each symbol name. In Source Insight, a syllable in a symbol or file name is considered a series of two or more characters that start with a capital letter. For example, the symbol name "CreateWindow" has two syllables: "Create" and "Window". Source Insight indexes both syllables so that you can browse for the symbol by typing "Cre" or "Win" or any combination, and in any order. Each syllable is prefix matched by what you type.

You can browse symbols this way using the Browse Project Symbols dialog box, and in the Project Window symbol and file list. This also works in the text box above the symbol window, on the left side of each source window.

In addition, all lists in Source Insight that are matched with a text box now have this standard ability as well.

What is a Symbol Syllable?

A syllable is a series of two or more characters that start with a capital letter.

A syllable in a symbol or file name is a series of two or more characters that start with a capital letter. It can also be a series of capital letters. Here are some examples:

Symbol Name	Syllables
CreateWindow	Create and Window
OpenHTML	Open and HTML
HTMLOpen	HTML and Open
FOpenDoc	Open and Doc
Vip32Test	Vip32 and Test

Symbol Indexes for Projects

Source Insight indexes all the symbols in the symbol database. Each symbol may appear in the indexes more than once. That is because there are three indexes that Source Insight maintains:

- **Full Name Index**. This indexes the full name of a symbol. The full name includes the symbol's parents. For example: "Class1.Member1". This type of index is always maintained and is used by Source Insight to navigate to symbol definitions.
- **Member Name Index.** This indexes only the structure and class member names of symbols. For example, a member function named "Help" inside of a class named "Document" would have a full name of "Document.Help", but the member name index will contain "Help". This index allows you to get to the member quickly, without specifying the class or structure it belongs to.
- **Syllable Index.** This indexes the syllables in a symbol's name. This index allows you to find symbols when you only know one or more parts of its name.

Index Options for Projects

The Project Settings dialog box specifies syllable and member indexing options.

Syllable and Member indexing takes up index space on disk and in memory. It also slows database access a little bit. You can control what is indexed in the Project Settings dialog box.

Two check boxes control indexing:

1. **Quick browsing for member names** – check this to allow member name indexing. When this option is enabled, you can simply type the member names of classes or structures, instead of having the type the class or structure name, followed by a dot (.) and the member

name. Un-checking this will save on disk space and memory.

2. **Quick browsing for symbol syllables** – check this to allow syllable indexing. When this option is enabled, you can type partial syllables and find symbols containing those syllables. Un-checking this will save on disk space and memory. If your project is large, then Source Insight may operate slowly when browsing, or synchronizing files with the symbol database.

If both check boxes are turned off, then the Browse All Symbols dialog box, and the Project Window symbol list filtering will revert to simple prefix matching. However, the Symbol Window on the left side of each source window, and the Project Window file list will still allow syllable browsing.

Controlling Syllable Matching

Enable and disable syllable matching in the Preferences: Typing dialog box.

You can control whether lists are affected by syllable matching in the Preferences: Typing dialog box. The check box: **Match syllables while typing** controls whether syllable matching is active or not, regardless of whether the project setting indicate that syllables are indexed in the symbol database.

Type a space character in front of a syllable to toggle syllable matching for an individual case.

As a shortcut, you can toggle the use of syllable matching by prefixing what you type with a space character. For example, if you turned syllable matching off in the Preferences: Typing dialog box, you can use it selectively by adding a space character in front of the syllable you type.

Using Syllable Matching

Type syllables into any text box associated with a list box.

Many of the text boxes where you can type a symbol or file name allow you to type a series of syllables. The list box associated with the text box will be filtered down based on what you type.

You can type multiple syllables for filtering.

When you type, begin with the starting two or more characters of a syllable, and follow that with another syllable. For example, "CreWin". You can separate syllables with a space also. For example, "Cre Win". For the most part, you can type the syllables in any order.

The syllable filtering is not case sensitive. However, once the filtered list is displayed, Source Insight will try to select the item in the list that most closely matches what you typed, including the case.

For example, the following specifications are equivalent:

```
CreWin
Cre Win
cre win
WinCre
Win Cre
win cre
win_cre
Win.cre
```

Matching names are sorted by how well they match.

Source Insight will filter the list contents down and sort the results, with best matches near the top. Symbols that match exactly what you typed, starting with the first character of the symbol, are displayed near the top of the list.

For example, if you have two symbols: "TopBottom" and "TopAndBottom", and you type "TopBot", then the "TopBottom" symbol will appear first, even though both symbol names match the specification.

Syllable Shortcuts

You can use these shortcuts when typing into a symbol name field to specify strict prefix matching, or member name matching:

- Prefix the spec with a single space or the caret (^) character to match strictly prefixes only. For example, "`^Format`" and "`<space>Format`" will match only symbols that start with "format". (Case insensitively.) It will not match "`LineFormat`". The prefix actually toggles syllable matching on and off. If you
- Prefix the spec with a dot (.) to match strictly prefixes of only leaf member names. For example, "`.fdirty`" will match **only** struct or class members that start with "`fdirty`". This also works with nested class members. For example, "`.draw`" will match the method "`Class1::Class2::Draw`".

Underscores, which are common in program identifier names, are ignored. For example, "_insert", and "insert" would also appear in the list together.

Accessing Symbol Information

Source Insight provides many ways to locate symbols, and review symbolic information in your projects. This is one of Source Insight's important strengths.

This section describes briefly the Source Insight features that help you access symbol information.

Background Parsing

The symbol database is automatically kept up-to-date.

While you edit your files, Source Insight scans your project files. This allows you to locate classes, methods, functions, etc., without having to compile your files.

Once a file has been added to your project, the names and locations of the symbol definitions in the file are stored in the project's symbol database. The symbol definition can be found quickly using the following techniques. Remember, you can right-click on many objects in Source Insight to bring up the object's shortcut menu. Some of the following commands are on the sh0rtcut menus.

Jump to Definition command

To jump to a definition, hold down Ctrl and double click on its name.

The Jump to Definition command jumps to the declaration of the symbol under the cursor. Type Alt+EQUAL or Ctrl+double-click with the left mouse button to invoke the command. You can also right-click on the symbol and use the shortcut menu to run this command.

After locating the symbol, the symbol's file is opened, displayed in a window, and the symbol name is selected. If you have more than one symbol with the same name in your project, Source Insight will ask you to select the one you want.

Jump to Caller command

The Jump to Caller command jumps to the caller of the function under the cursor. You can right-click on the symbol and use the shortcut menu to run this command. This only works if the object under the cursor is a function name.

Browse Project Symbols command

The Browse Project Symbols dialog box lists all the symbols in your project.

The Browse Project Symbols command opens a dialog box containing a list of all symbols in the project. You can select a symbol from the list, or you can type in a symbol name. When the dialog box first appears, the word under the cursor is automatically loaded into the Symbol text box.

When you start typing into the Browse… dialog boxes, the list is automatically filtered down to match what you've typed. This makes it easy to get to a given symbol, usually by typing only a few characters.

Browse Local File Symbols command

The Browse Local File Symbols command opens a dialog box containing a list of all symbols found in the current file only. This command works even if you do not have a project open.

Project Window Symbol List

The Project Window also lists all the symbols in your project, and it is modeless.

The Project Window displays a list of all project symbols. Like the Browse Project Symbols dialog box, you can type into the text box to perform prefix and syllable name matching. Unlike the Browse Project Symbols dialog box, the Project Window is a modeless window. Therefore, as you select symbols in the Project Window, the Context Window and Relation Window will update and provide information about the selected symbol.

The Project Window also has a class view. The class view is a hierarchical view, where symbols are listed by category. For example, all global variables are grouped together; all structs are grouped together, and so on. For structured types, such as classes, structs, and unions, the Project Window lets you expand them to show their members.

Call Trees and Reference Trees

You can view call trees and references trees with the Relation Window.

The Relation Window is another Source Insight innovation that shows the relationship between the currently selected symbol and other things. It works like the Context Window by tracking what you are doing and showing relationship information automatically.

The Relation Window can show function call trees, class hierarchies, structure members, reference trees, and more.

Context Window

The Context Window shows the definition of the selected symbol.

The Context Window is a Source Insight innovation introduced in version 2.0. It provides symbolic information automatically. It tracks what you are selecting and typing and shows you relevant symbol declarations automatically, while you work.

Command Line

You can jump to a symbol by using the `-f <symbol>` option on the Source Insight command line when you start Source Insight.

For example:

```
insight3 -f myfunc
```

This will position to the symbol named "myfunc".

You can also simply give the name of the symbol on the command line without a special command option, as though it were a file name. In this case, Source Insight will try to determine whether the symbol you specified is either a file, or a parsed symbol. You can list as many symbols on the command line as you like, just as with file names.

For example:

```
insight3 myfunc fcb init
```

This will open all files where the symbols "myfunc", "fcb", and "init" are located. See "The Command Line" on page 145.

Finding References to Symbols

Use Lookup References to find references to symbols quickly.

The Lookup References command can be used to quickly find all textual references to the word in the current selection. For example, to find all calls to TextOut, put the insertion point inside of the TextOut word and run the Lookup References command. A Search Results window will be built which will list all references found.

The Browse Project Symbols, Browse Local File Symbols, and Symbol Info dialog boxes also have "References" buttons, which do the same thing. See "Lookup References" on page 272.

Project Report

The Project Report command will create a report file containing a symbol cross-reference, and statistics about the files in your project.

Smart Renaming

Use Smart Rename to perform context-sensitive renaming across project files.

The Smart Rename command is a context-sensitive form of a global search & replace. It renames an identifier across all project files using a smart context-sensitive method. Source Insight's search index makes the search very fast. This is the easiest way to replace a single word identifier with a new string. In addition, you can have Source Insight produce a log of replacements in the Search Results window. Each replacement line is listed, along with a source link to the location of each line that was changed.

Syntax Formatting and Styles

Syntax Formatting is an important Source Insight feature that renders information in a dense, yet pleasing and useful way.

Syntax Formatting uses rich text formatting based on program information.

Source Insight uses information gathered from its parsers to format source code. Identifiers can be displayed in different fonts or font sizes, along with a variety of effects such as bold and italics.

Formatting is applied with "styles". A style is a set of formatting properties. For example, a style may specify bold + italic. You can edit each style's formatting properties with the Style Properties command.

Formatting *styles* are applied to source code elements.

There are several pre-defined styles. You can also add your own. The styles are stored in the configuration file.

How a Style Works

Styles contain formatting instructions that can be combined with other styles.

When a style is applied to text, it combines with the underlying text properties. The basic text properties are set in each document type's font settings. You can control this with the Options > Document Options command.

Formatting Properties

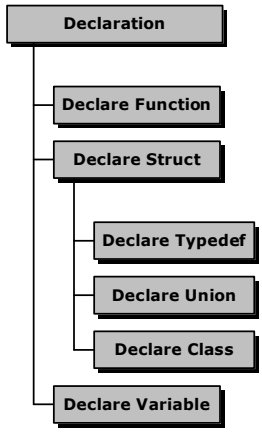
Each style is like a list of formatting differences from its parent style. For example if a style has the "bold" property, then "bold" is added to the parent. Each formatting item has the following possible states:

- **On**. The formatting property is added. E.g. Bold-On
- **Off**. The formatting property is removed. E.g. Bold-Off
- **A Number.** This number applies to items like scaling, or font point size. E.g. scaling = 120%.
- **Font Name.** This applies to the Font Name item.
- **No Change**. The style has no effect on the formatting property. It inherits the properties of the parent style.

Parent Styles

Styles are organized into a hierarchy.

Styles have a "Parent Style" property. Styles inherit their formatting properties from their parent style. This allows you to create a hierarchy of styles. For example, the built-in styles contain a hierarchy for Declarations, a portion of which looks like this:



Style formatting properties combine with the parent style.

Formatting properties in a style are combined with the parent style. Thus, the Declare Struct style inherits the formatting properties of the Declaration style. That lets you affect changes to all declaration styles by altering the single Declaration style.

The topmost parent style is the "Default Text" style. Its formatting properties are determined by the document type's font settings.

How Styles Apply to Source Code

Styles are automatically applied to source code text.

Styles are automatically applied to source code text. With the exception of comment styles, you cannot explicitly apply a style yourself, like in a word processor.

Language Keywords

You can associate your own keywords with any style.

The simplest application of styles is for formatting language keywords. Each language contains a keyword list. Each keyword list associates a keyword, for example "`while`", with a style. The keyword list is editable and is stored in the configuration file. Source Insight will recognize a keyword, and apply the associated style to the keyword. You edit the language keyword list in the Preferences: Languages dialog box.

Declarations

Declarations are formatted with a "Declare…" style.

Wherever a symbol is declared or defined, its name is formatted with an appropriate "Declaration" style. There are declaration styles for different types of things, like structs, classes, unions, functions, etc.

For example, a structure might look like this:

```
typedef struct CLIP
    {
    HSZ         hszSource;      // where it came from
    HSZ         hszWhat;        // what it is
    } CLIP;
```

The structure name "`CLIP`" is shown in the "Declare Struct" style. The member fields inside the struct are shown in the "Declare Member" style.

References to Identifiers

References to symbols are formatted with a "Ref to…" style.

References to symbols that are not actual declarations are formatted with an appropriate "Reference" style. There are reference styles for different types of symbols.

For example, a code fragment might look like:

```
// seek the class name token
SeekTokenLnIch(hpar, HdocOfTgl(htglLocal), kswaChangeMark);
```

The call to "`SeekTokenLnIch`" is formatted with the "Ref to Function" style. The C macro function "`HdocOfTgl`" is formatted with the "Ref to Macro" style, which helps up know just by looking at it that it is a macro. The parameter "`hpar`" is formatted with the

"Ref to Parameter" style, so we know it is a local parameter of the current function. The identifier "`htglLocal`" is formatted with the "Ref to Local Variable" style, so we know it is a local variable in the current function. The "`kswaChangeMark`" constant is formatted with the "Ref to Constant" style.

Reference styles are very useful, but consume some processing power.

Reference formatting gives you a lot of information without having to ask for it. It becomes instantly obvious if you have misspelled a function name, or whether you are using a constant, or a local variable, or a global variable.

You should be aware that enabling reference formatting can slow the display down in some cases. Source Insight needs to perform symbol lookup operation each time it encounters a potential symbol reference.

Inactive Code

Source Insight's C/C++ and Resource File parsers can be made to recognize inactive blocks of code that are disabled at compile-time with `#ifdef` directives. The Options > Preferences: Language dialog box has "Conditions" buttons that allows you to edit the list of known conditional constants.

Inactive code is formatted with the Inactive Code style.

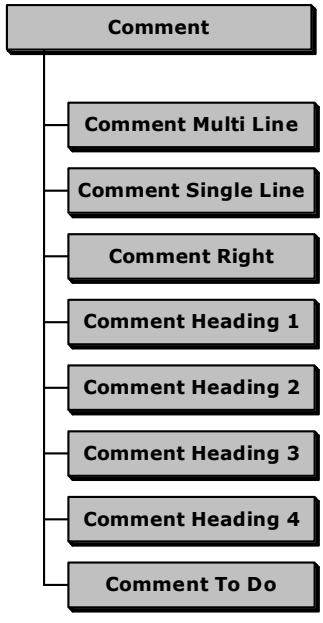
```
#ifdef NEVER
        TluParentRefFromSrl(hsrl, ist, &tlu);
        tlu.hdoc = hdocNil;
#endif /* NEVER */
        return CmdGotoTlu(&tlu) == cmdOK;
```

Blocks of code that are inactive are given the "Inactive Code" style.

For more information, see Inactive Code - ifdef Support.

Comment Styles

Source Insight allows you to add some explicit formatting to comments. The comment style hierarchy appears below.



Comment Headings

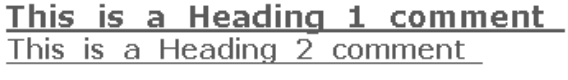
Comment heading styles explicitly format comments.

Comment heading styles are very useful for breaking up large chunks of code with high-level comments.

Comment Heading styles are specified with a `//n` comment, where `n` is a number between 1 and 4. For example:

```
//1 This is a Heading 1 comment
//2 This is a Heading 2 comment
```

When the comment displays, the `//n` at the beginning of the comment is hidden, unless the cursor is on that line.



If the cursor is on the line, then the `//n` will appear so that you can edit it.

Comment Right

Comments that appear to the right of code are given the "Comment Right" style. For example:

```
// this comment is formatted with the Comment Line style
a = 0;  // this is formatted with the Comment Right style
```

Single and Multi Line Comments

Multi-Line comments are displayed in the Comment Multi Line style. These include any comments using the `/*` and `*/` delimiters in C/C++ and Java.

Single line comments are displayed in the Comment Line style. These include comments that use the `//` delimiters in C/C++ and Java.

Comment Styles and Custom Languages

When you define a custom language, using the Preferences: Languages dialog box, you can specify comment or text-range types, and associate them with a style. For example, you could create a comment range that starts with `(*` and ends with `*)` and the text is formatted with the "Comment Multi Line" style. You can actually define any delimited range of text, and associate it with any style (including a non-comment style). See "Special Language Options" on page 256.

Syntax Decorations

Syntax Decorations add extra information to your code display.

Source Insight can replace some common operators with more useful symbolic characters. The Preferences: Syntax Decorations command lets you control which decorations are used.

The symbolic characters are formatted with the "Symbol Characters" style. If you use the Style Properties command to look at the Symbol Characters style, you will see that it uses the Symbol font. You may change the style's properties, such as the color or font size, but if you change the font name to something other than Symbol, your syntax decoration symbols will probably not show up correctly.

Note: It's important to remember that symbol substitutions do not change the text in the source file; only its representation on the screen changes to show the special symbols. You still need to type the operators normally when editing your code, or when searching for them.

Operator Substitutions

You can display operators with special symbols.

Common operators, such as the pointer de-reference right arrow (->), or the assignment operator (=) can be replaced with symbolic operators, such as arrows. For example, instead of

```
DIM dimLine = DimOfRgch(hdc, psnoMem->tav.sz, 1);

DrawNodeLine(psnoMem->tav.sz, hdc, xp, yp, prc);
```

With decorations on, the = and -> are replaced with real arrows:

```
DIM dimLine ← DimOfRgch(hdc, psnoMem→tav.sz, 1);

DrawNodeLine(psnoMem→tav.sz, hdc, xp, yp, prc);
```

Boolean, math, and other operators can be substituted with decorative symbols too.

Scaled Nested Parentheses

Source Insight can display nested parentheses in different font sizes to make it easier to identify matching sets. This even works across multiple lines.

```
i ← (ITIR) (((UINT)iMin + (UINT)iLim) >> 1);
```

Goto Arrows

Another useful decoration is the "goto arrow". An up or down arrow appears in `goto` statements which points in the direction of the target label.

```
krel ← krelCalls;
goto ↑LSet;
```

End Brace Annotations

Source Insight can also add automatic "end brace" annotations to the closing curly brace in C/C++ and Java code. This makes it easier to deal with nested if, while, switch, and other blocks of code.

```
        } « end switch *pch »
    } « end if stcNewWord!=stcString... »
} « end if !FWhiteCh(*pch) »  // !fWhiteCh
```

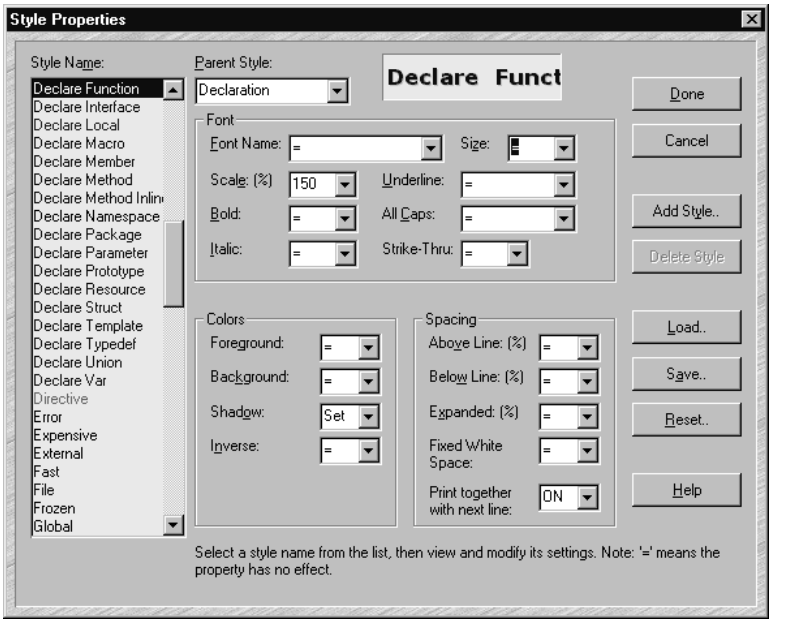
Controlling Syntax Formatting

You have great control over what and how Source Insight formats your code.

Style Properties

Change style formatting with the Style Properties command.

The Style Properties command lets you control the formatting properties of each style. You can also add your own styles, and import and export them. For more information, see "Style Properties" on page 346.



The Syntax Formatting Command

Control what how styles are used in the Syntax Formatting dialog box.

The Preferences: Syntax Formatting command lets you control how much display formatting is applied. You can speed up the display by disabling some options here. For more information, see "Syntax Formatting" on page 359.

The Syntax Decorations Command

This Preferences: Syntax Decorations command lets you control what syntax decorations are performed.

Turning Off Syntax Formatting

The Syntax Formatting features of Source Insight are powerful, but sometimes you need to see how text will line up in another editor or in a simple display mode when only a single font is used.

Switch Off Syntax Formatting Temporarily

Use Draft View to temporarily see your files without syntax formatting.

If you want to temporarily see your files without syntax formatting, use the View > Draft View command. Draft mode is useful for quickly switching your display to a basic monospaced font display. This is particularly useful if you are using spaces instead of tab characters to line columns up. When draft mode is active, it overrides the settings of the Preferences: Syntax Formatting and Syntax Decorations dialog boxes.

I Don't Want Fonts to Change

If you want to just see only color formatting, but no font changes, or font embellishments such as bold and italic, then use the Options > Preferences: Syntax Formatting dialog box and check the **Use only color formatting** box.

I Want All Characters to Have the Same Width

If you want to permanently see your files with all characters the same width, then right-click on your file and select Document Options. Click the **Screen Fonts** button and pick a monospaced font, such as Courier New. Of course, you can pick any font you want to change the default font used for that document type.

Selection Shortcuts

Source Insight provides many short cuts for selecting meaningful objects within your source code.

Source Insight has a "selection bar" area in the left margin of each source file window. Many shortcuts involve clicking in the selection bar.

Selecting Whole Words

To select by whole words, double-click on them.

Simply double-click on a word to select the whole word. You can drag out a selection in this mode to select whole words.

Selecting Whole Functions or Symbols

To select a whole function, double-click in the left margin next to it.

To select a whole function, struct, or other type of symbol, simply double-click in the selection bar area of a source file window. If you hold the mouse button down and drag, Source Insight will select whole functions or symbols as you drag.

You can also use the Select Symbol command to select the symbol that encloses the current text selection.

The Symbol Window (on the left of the source file window) also can be used to select whole symbols. Double-clicking on any entry in the list will select the symbol.

Selecting Matching Parentheses and Blocks

To select to the matching parentheses, brace, or quote, just double click on it, or use Alt+EQUAL.

To select to the matching brace or parentheses, double-click on any parentheses, square brace, curly brace, or double-quote mark. Source Insight selects up to its match. The Select Match command (Alt+EQUAL) also selects up to the matching brace or parentheses.

Selecting the Enclosing Block

The Select Block command selects the enclosing block. An enclosing block is one that is surrounded by either a parentheses, square

brace, or curly brace. If you repeat this command, it will select the next outer-most block.

Selecting a Whole Line

To select a whole line of text, simply click in the selection bar. A whole line of text will be selected. If you hold the mouse button down and drag, whole lines will be selected.

Selecting a Paragraph of Text

For files that do not have a language parser attached (See Document Options), you can double-click in the selection bar to select a whole paragraph of text.

A paragraph is defined as a group of text lines surrounded by blank lines.

Selecting Between Lines

Source Insight allows you to select in between lines by providing a wedge shaped cursor when the cursor is near the start of a line and slightly above or below it. When you click at that point with the mouse, Source Insight selects the space between the lines. If you start typing, Source Insight automatically inserts a line break first. This is a handy way to insert a new line.

Searching and Replacing Text

Source Insight provides a number of searching commands, and replacement commands that operate on the current file as well as across multiple files. When searching multiple files, source links can be used to quickly link matches.

Searching is an import activity for programmers. As such, Source Insight devotes a lot technology to help you wade through copious source code.

Searching for Symbol References

Use Lookup References to find context-sensitive references to symbols.

To search the current project for references to the currently selected symbol, use the Lookup References command (Ctrl+/). For example, click inside "BeginPaint", run the Lookup References command, and Source Insight will open a Search Results window, which lists all the places you used BeginPaint in your project. Each matching line listed in the Search Results window also has a source link to the location of each line containing the word you looked up.

The Lookup References command is context sensitive, so that if finds only the correctly matching references, given the scoping context.

Renaming an Identifier

Use Smart Rename to perform context-sensitive renaming of symbols - global or local.

To rename an identifier across all project files using a smart context-sensitive method, use the Smart Rename command (Ctrl+Single-Quote). Smart Rename is a context-sensitive form of a global Search and Replace. Source Insight's search index makes the search very fast. This is the easiest way to replace a single word identifier with a new string. In addition, you can have Source Insight produce a log of replacements in the Search Results window. Each replacement line is listed, along with a source link to the location of each line that was changed.

Smart Rename is also excellent for renaming local scope variables.

Searching the Current File

The Search command searches the current file buffer.

The Search command (Alt+F) searches the current file for a search pattern. The search can be forward or backwards. In addition, the search can be batch style, where the search output results are placed in a Search Results window.

The Search Forward (F4) command repeats the last search, allowing you to advance through the file, finding each occurrence of a pattern.

The Search Forward for Selection command searches for the next occurrence of the first word in the current selection.

The Incremental Search command (F12) finds matches as you type the characters.

Replacing in the Current File

The Replace command searches the current file for a search pattern, and replaces the pattern with a new pattern. The range of replacement can be the whole file, or just the current selection.

Searching Multiple Files

The Search Files command searches across multiple files.

To search for a pattern across all project files, or other non-project files, use the Search Files command. The Search Files command is similar to the Search command, except you can specify what files you want to search. The results of the search are placed in the Search Results window. The search results also contain hidden information called source links.

If you are just looking for single, whole-word references across your whole project, the Lookup References command is much faster.

Replacing in Multiple Files

The Replace Files command replaces across multiple files.

The Replace Files command is similar to the Replace command, except you can specify what files you want to do the replaces in.

If you want to rename a symbol across multiple files, the Smart Rename command is usually better, because it is context sensitive.

Searching for Keywords

The Search Project command can do an Internet-style search on your project.

If you want to search your whole project as though it was an Internet web site, then use the Search Project command. Using keyword searching, you can find any combination of terms that occur within a specified number of lines of context.

Regular Expressions

Regular expressions are special search strings that are useful for matching complicated patterns. In a regular expression string, many characters have special meanings. For example, there is a special character that means "the beginning of the line". This section describes all the special characters understood by Source Insight.

Wildcard Matching

```
. (dot)
```

The dot `.` matches any character.

Example: `b.g` matches `big`, `beg`, and `bag`, but not `bp` or `baag`.

Matching the Beginning or End of a Line

```
^ and $
```

The caret `^` matches the beginning of a line when the caret appears as the first character in the search pattern.

Example: `^Hello` matches only if `Hello` appears at the beginning of a line.

The `$` matches the end of a line.

Example: `TRUE$` matches only if `TRUE` appears at the very end of a line.

Matching a Tab or Space

```
\t
\s
\w
```

`\t` matches a single tab character.

Example: `\tint abc;` matches a tab character followed by `int abc;`.

`\s` matches a single space character.

Example: `\sif` matches a space character followed by `if`.

\w matches a single white space character. In other words, \w matches either a tab or space character.

Example: \wwhile matches either a tab or space character, followed by while.

Matching 0, 1, or More Occurrences

```
* and +
```

* matches zero or more occurrences of the preceding character. The fewest possible occurrences of a pattern will satisfy the match.

Example: a*b will match b, ab, aab, aaab, aaaab, and so on.

+ matches one or more occurrences of the preceding character.

Example: a+b will match ab, aab, aaab, aaaab, and so on, but not just b.

Matching Any in a Set of Characters

```
[ .. ]
```

When a list of characters are enclosed in square braces [..] then any character in that set will be matched.

Example: [abc] matches a, b, and c, but not d.

When a caret ^ appears at the beginning of the set, the match succeeds only if the character is *not* in the set.

Example: [^abc] matches d, e, or f, but not a, b, or c.

Sets can conveniently be described with a *range*. A range is specified by two characters separated by a dash, such as [a-z]. The beginning character must have a lower ASCII value than the ending character.

Example: [a-z] matches any character in the range a through z, but not A or 1 or 2.

Sets can contain multiple ranges.

Example 1: [a-zA-Z] matches any alphabetic character.

Example 2: [^a-zA-Z0-9] matches any non-alphanumeric character.

Regular Expression Groups

`\( and \)`

Parts of a regular expression can be isolated by enclosing them with `\(` and `\)`, thereby forming a *group*. Groups are useful for extracting part of a match to be used in a replacement pattern. Each group in a pattern is assigned a number, starting with 1, from left to right.

Example: `abc\(xyz\)` matches `abcxyz`. `xyz` is considered group #1.

This is not all that useful, unless we are using the Replace command. The replace string can contain group characters in the form of `\<number>`. Each time a group character is encountered in the replacement pattern, it means "substitute the group value from the matched pattern".

Example 1: replace `\(abc\)\(xyz\)` with `\2\1`. This replaces the matched string `abcxyz` with the contents of group #2 `xyz`, followed by the contents of group #1 `abc`. So `abcxyz` is replaced with `xyzabc`. This is still not too amazing. See the next example.

Example 2: replace `\(\w+\)\(.*\)ing` with `\1\2ed`. This changes words ending in `ing` with the same word ending with `ed`. Your English teacher would not be too happy.

Overriding Regular Expression Characters

`\ (backslash)`

A backslash character `\` preceding a meta-character overrides its special meaning. The backslash is ignored from the string.

Example: `a\*b` matches `a*b` literally. The `*` character does not mean "match 0 or more occurrences".

Regular Expression Summary

The following special characters are interpreted in regular expressions

Character	Matches
^ (at the beginning only)	beginning of line
.	any single character
[abc]	any single character that belongs to the set abc
[^abc]	any single character that does not belong to the set abc
*	zero or more occurrences of the preceding character
+	one or more occurrences of the preceding character
\t	a tab character
\s	a space character
\w	white space (a tab or a space character)
$	the end of the line

Sets, such as [abc] may be in the following formats.

Set type	Meaning
[<character list>] eg. [abcde]	Matches any character within the set. The set can be any number of characters long.
[x-y] eg. [a-z]	Matches on any character within the range of x through y, inclusively. The ASCII value of x must be less than that of y.
combination; eg. [WXYa-z0-9]	Character lists and ranges may be combined.

Bookmarks

Bookmarks mark a particular location in a file. The Bookmark command allows you to set, jump to, and remove marks from the mark list. Each mark has a name, and specifies a file and line number. The Bookmark command also displays the function or symbol where the mark is located

Bookmarks are useful for keeping track of particular locations of interest in your files. The marks you have set are saved in the current workspace, so they will be preserved from session to session.

Bookmarks are maintained as you edit your files. For example, if you set a mark on a particular line, and then insert lines before that one, the mark will still be on the original line of text, even though the actual line number may have changed. If the line that the mark is set at is deleted or the file is closed, then the mark is deleted also.

Selection History

The *selection history* is a circular list of your last 100 selection positions in the currently opened files.

The Selection History is a circular list of recently visited position.

You can use the Selection History command to display and jump to positions in the selection history. The Selection History command also displays the function or symbol where each history item is located, along with its file and line number.

In this example, a function call chain is in the selection history. WinMain calls InitApp which calls FStartDisp which calls HplAlloc. The last (i.e. "current") position is #4.

1. WinMain in Main.c at line 1012
2. InitApp in Init.c at line 103
3. FStartDisp in Disp.c at line 201
4. >> HplAlloc in Plex.c at line 421

Go Back and Go Forward commands

Use the Go Back and Go Forward commands like you would in an Internet browser.

The Go Back and Jump To Definition commands make it easy to travel down a function call chain, and "pop" back up the chain, and then travel down another path with a different function call.

The Go Back command jumps to the previous location in the selection history. The Go Forward command jumps to the next location in the selection history. The selection history list is circular, so if you reach the end of the list, the Go Forward and Go Back commands will wrap around to the other end of the list.

If you know how to use the Forward and Back buttons in an Internet browser, then you should understand the Go Back and Go Forwards commands completely.

The Go Back command is like popping selection history elements off of a stack. It "backs up" the selection history "pointer". In this example, the pointer is backed up to position #3.

1. WinMain in Main.c at line 1012
2. InitApp in Init.c at line 103
3. >> FStartDisp in Disp.c at line 201
4. HplAlloc in Plex.c at line 421

As you move on to new locations, the selection history is written over starting at the "current pointer" location. After using the Go Back command, using the Jump To Definition command writes a new element to the selection history list.

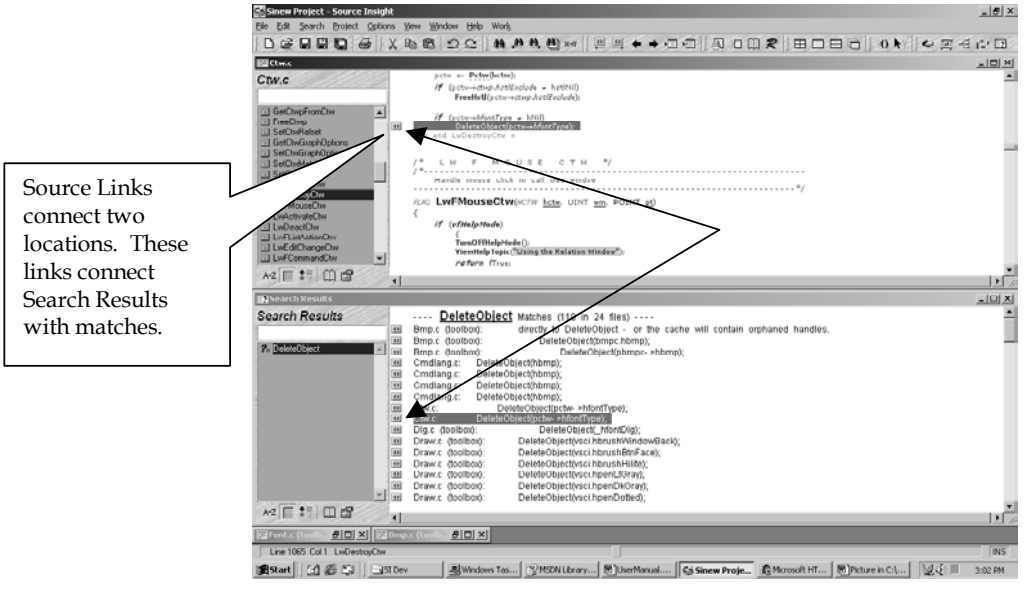
1. WinMain in Main.c at line 1012
2. InitApp in Init.c at line 103
3. FStartDisp in Disp.c at line 201
4. >> GetWinSize in Win.c at line 53

Source Links

Source Links connect two locations in two different text files. They connect a line of text in a "link source" file to a location in a "link target" file. Links are associated with individual lines. Source Links are part of the current workspace.

Source Links connect two lines in two files.

Links are traversed by using the Jump To Link command, which takes you to the other end of the source link at the current line. A link can be traversed as long as the link source file is open. If the link target file is not open, the Jump To Link command will open the file automatically.



Source Links are bi-directional, so you can use the Jump To Link command to go from the link source to the target, or from the target back to the source.

In addition, the link information is maintained as you edit your files, just as bookmarks are. A link is only removed if you delete its link source line, or close the link source file.

Searching and Source Links

The search results placed in the Search Results window by the Search, Search Files, and Lookup References commands contain Source Links. The link source file is the Search Results temporary

file, and the link target for each link is the location of the matching patterns in the various files that were searched.

Source links are part of the current workspace, which means it is part of the session state. If you save the Search Results to a new file name, the Source links will be retained in the current workspace. For example, you could search for one pattern, save the Search Results to S1.OUT, do a second search, and both S1.OUT and Search Results will have their source links. If you close the Search Results window, the source links are deleted.

Creating Source Links

The Parse Source Links creates source links in the current file.

You can use the Parse Source Links command to create source links in the current file. The Parse Source Links command requires that you specify a search pattern to be used to parse file names and line numbers from a file. Each time a pattern matches, a new source link is inserted into the current file.

The Parse Source Links command is useful if you have a "log" file that contains compiler output and error messages. You just open the log file, and run the Parse Source Links command. A link will be setup for each line in the log file containing an error message.

Source Links from Custom Command Output

Custom command output can contain source links for errors.

You can create source links from the output of a custom command. When defining the command, the "Parse Source Links" option should be on. You must specify a search pattern to be used to parse file names and line numbers. When the command terminates, Source Insight automatically invokes the Go To First Link command.

This is the recommended way to invoke the C compiler from Source Insight. This allows you to run the compiler, and if there any errors, Source Insight will position you to each erroneous C source line.

Navigating with Source Links

Source Insight provides the following commands for moving between link locations.

Command	Key	Description
Jump To Link	Alt+Up Arrow	Moves to the other end of the source link at the current line.
Go To First Link	Shift+F8	Selects the first link line in the link source file, and selects the associated link line in the link target file, and ensures both files are visible on the screen in windows.
Go To Next Link	Shift+F9	Same as above, except with the next link in the link source file.
Go To Previous Link		Jumps to the next link in the source file.

Scrolling and Selecting Text

A *selection* is a range of zero or more characters highlighted in a source file window. Selected text is highlighted displayed in the "Selection" style; usually reverse video.

Each source file window has its own single selection. Switching between windows leaves the selections in each window intact.

Selections can be made with the keyboard and the mouse. Commands used for selecting text are listed below. To select text with the mouse, simply point at the text and drag across it. Double-clicking the left mouse button selects a whole word.

Scrolling the window will only change the selection if the "Keep cursor in window when paging up and down" option is turned on in the Preferences: General dialog box. Otherwise, only cursor movement keys, pointing with the mouse, or editing your file will change the selection.

The selection in the current window and current file is referred to as the *current selection*.

Moving Through a File

There are many commands to move you around in a file. It's important to know that there are two types of movement: *selecting* and *scrolling*.

Selecting is moving the current selection (usually an insertion point) around in the file.

Scrolling is when the file window is scrolled to reveal new parts of the file. You can scroll a window up, down, left, and right. Scrolling does not always affect the location of the selection.

Selection commands can cause scrolling to occur if the place you are selecting is not visible in the window. For example, if the cursor is on the bottom line of a window and use the Cursor Down command, the window is scrolled to reveal the line below.

Using the Scrolling Commands

The scrolling commands scroll the active window. They do not affect the current selection. They only affect the window.

Command	Key	Description
Scroll Line Up	Alt+Up	Scrolls up by one line
Scroll Line Down	Alt+Down	Scrolls down by one line
Page Up	PgUp	Scrolls up by a window full
Page Down	PgDn	Scrolls down by a window full
Scroll Half Page Up	Ctrl+PgUp	Scrolls up by half a window
Scroll Half Page Down	Ctrl+PgDn	Scrolls down by half a window
Scroll Left	Alt+Left	Scrolls left by aprox. 1 tab stop
Scroll Right	Alt+Right	Scrolls right by aprox. 1 tab stop

The scroll bar

Of course, you can use the scroll bar with the mouse to scroll around in a file. The scroll bar mimics the following commands.

Command	Scroll bar usage
Scroll Line Up	Click in the Up Arrow
Scroll Line Down	Click in the Down Arrow
Page Up	Click above the scroll bar thumb
Page Down	Click below the scroll bar thumb

Using the Selection Commands

The selection commands change the current selection. Usually, if the resulting selection is not visible in the window, the window is scrolled to show it. These commands change the selection to an insertion point and move it to a new location in the file.

Command	Key	Description
Cursor Down	Down Arrow	Move down by one line.
Cursor Up	Up Arrow	Move up by one line.
Cursor Left	Left Arrow	Move left by one character.
Cursor Right	Right Arrow	Move right by one character.
Beginning of Line	Home	Move to beginning of line.
End of Line	End	Move to end of line.
Top of File	Ctrl+Home	Move to top of file.
Bottom of File	Ctrl+End	Move to end of file.
Beginning of Selection	Ctrl+Alt+[	Move to start of an extended selection.
End of Selection	Ctrl+Alt+]	Move to end of an extended selection.
Top of Window		Move to top of window.
Bottom of Window		Move to bottom of window.
Word Left	Ctrl+Left	Move left by one word.
Word Right	Ctrl+Right	Move right by one word.
Function Down	Keypad +	Move to next function definition.
Function Up	Keypad -	Move to previous function definition.
Blank Line Down		Move to next blank line.
Blank Line Up		Move to previous blank line.
Paren Left	Ctrl+9	Move to previous enclosing parentheses.
Paren Right	Ctrl+0	Move to next enclosing parentheses.
Blank Line Up		Move to previous blank line.
Blank Line Up		Move to previous blank line.

Block Up	Ctrl+Shift+{	Move to previous { block level.
Block Down	Ctrl+Shift+}	Move to next { block level.
Go To Line	F5, Ctrl+G	Move to a specified line number.
Search	Ctrl+F	Search for occurrence of a pattern.
Search Forward	F4	Search for next occurrence.
Search Backward	F3	Search for previous occurrence.
Search Forward for Selection	Shfit+F4	Searches for next occurrence of the word under the cursor.
Selection History	Ctrl+Shift+M	Displays a list of your past selections.

Extending the selection

These commands will create an extended selection or extend an existing selection. The keystrokes listed are the defaults.

Command	Key	Description
Select All	Hold down Ctrl and click the mouse in the selection bar at the left.	Select the whole file.
Select Block	Ctrl+-	Selects the next smallest enclosing set of braces or parentheses.
Select Function Or Symbol	Double-click in selection bar at the left of edge of the window	Select the whole enclosing symbol definition.
Select Char Left	Shift+Left	Extends selection to the left by one character.
Select Char Right	Shift+Right	Extends selection to the right by one character.
Select Line	Shift+F6	Selects whole line.
Select Line Down	Shift+Down	Extends selection down a line.
Select Line Up	Shift+Up	Extends selection up by a line.
Select Match	Alt+=	Selects up to the matching brace, parentheses, or quote mark.
Select Sentence	Shift+F7, Ctrl+.	Selects whole sentence (up to the next period).
Select Word	Shift+F5, and Double-click L. Mouse	Selects whole word.
Select Word Left	Ctrl+Shift+Left	Extends selection to include the whole word on the to the left.
Select Word Right	Ctrl+Shift+Right	Extends selection to include the whole word on the to the Right.
Select To	Shift+L. Mouse	Extends the selection to the current mouse location. This is assigned to the mouse.

Select To Top Of File	Shift+Home	Selects up to the beginning of the file.
Select To End Of File	Shift+End	Selects up to the end of the file.
Select To End Of Line		Selects up the end of the current line.
Toggle Extend Mode		When on, movement keys extend the selection.

The Toggle Extend Mode command toggles Extend Mode on and off. When Extend Mode is on and you use a movement command, such as Cursor Left, the current selection is extended in that direction.

See Also: Finding Symbols in a Project.

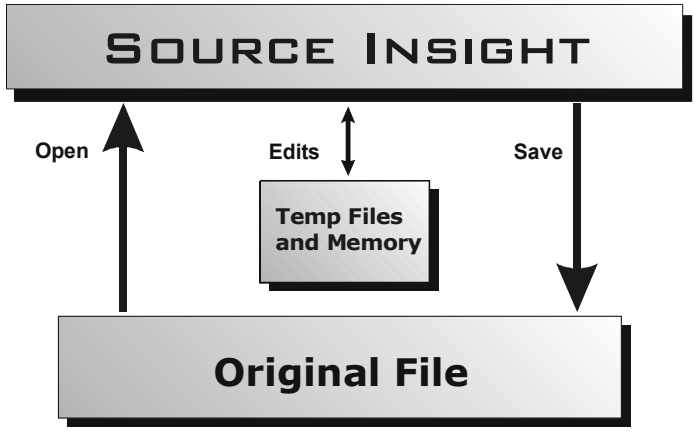
How Source Insight Edits Files

The Open command sucks a file into a working "buffer".

The Open command opens a file on disk and loads it into a file buffer. A *file buffer* is the temporary image of the file that you can edit. You can edit the file buffer without affecting the original file, until you save it using the Save command.

The Save command overwrites the original file.

The Save command writes the file buffer contents back over the original file. This is the only time the original file is changed.



When you close the file, the file buffer contents are simply thrown away. The original file on disk is not changed unless you use the Save command first.

The original file is untouched until you save the file.

All changes are recorded in the temporary file buffer until the file is saved. When the file is saved, a new file is created and written and the original file is either deleted, or optionally moved to the backup directory. Once the new file is completely and safely written to disk, the file is renamed back to the original file name. By not altering the original file until it is saved, Source Insight provides a safe mechanism for modifying files.

It is helpful to think of using a file buffer when editing a file, however throughout this documentation, when references are made to an “open file”, it actually means a file buffer. Source Insight maintains a close connection between an open file and its original, base file. Therefore, you can think of the Open command as a command that opens the original file so that you can edit it, rather than a command that copies the original file into an abstract buffer that exists independently of any files. You can think of the Save command as synchronizing the original version with the edited version.

Source Insight is used to edit text files only.

Source Insight is used to edit ASCII text files. Source Insight was not designed to edit non-ASCII files. There is nothing to stop Source Insight from opening such a file, however. In the event that a file you open contains non-ASCII characters, Source Insight will not allow you to save the file without first asking if you are sure you want to. You usually do not want to do so, since Source Insight treats CR/LF sequences as end of line markers, and will always make sure a saved file contains a CR/LF sequence at the end.

You can also open virtually any type of file by dragging a file and dropping it onto the Source Insight application window. This includes project files (.PR), configuration files (.CF3), workspace files (.WK3), and clip files (.CLI), in addition to ordinary text files.

Time stamping

A file's time stamp and size are used as its "signature".

When you save a file with Source Insight and you have a project open, Source Insight records the file's modification time in the project. If you open a project file and the file's modification date is newer than the date recorded in the project, then Source Insight will re-synchronize the file project symbol database for that file. By automatically synchronizing, Source Insight allows you or others in your development team to use another text editor or a source control system and still maintain Source Insight's project databases.
See Also: Synchronize Files.

What Happens when you Start Source Insight

When you start Source Insight, it first looks in the Registry to see what project needs to be opened. Then, it opens that project, thereby loading that project's configuration and workspace. Loading a workspace causes all the previously opened files to be reopened.

If file names are specified on the command line, then the files listed in the workspace are usually not reopened.

If a project is opened, the current working directory may be changed to the project's root directory.

See Also: Recovering From Crashes.

Recovering From Crashes

The recovery file is saved in the background, while you edit.

Periodically, your unsaved edits are saved in a temporary recovery file. This saving process is very fast and you will usually not notice that anything is being done. You will not be interrupted.

The recovery file actually only contains pointers to other files. Therefore, it is small and fast to write out. By saving edits incrementally to the recovery file, Source Insight insures you against a system failure, power outage, or crashes. Source Insight will recognize when a recovery needs to be done the next time you run it.

You control how often the recovery file is saved in Preferences: General.

The Preferences: General command allows you to set how often the recovery file is synchronized. You can specify this recovery time in seconds. The default value is every 15 seconds. If you make the recovery time smaller, the recovery file is updated more often. If you make it larger, the recovery file is updated less often. If you make it too long, you will be able to do many changes to your files without the changes being recorded in the recovery file. You should the keep the period as short as possible. However, the more frequently the recovery file is updated, the slower Source Insight's performance becomes. However, on fast machines, it is not noticeable, so you might as well make it very short (e.g. 5 sec).

Recovery Procedure

An orphaned recovery file signals that a crash occurred.

If you have been editing and a crash occurred before you could save your files, then simply run Source Insight again. Source Insight will know that there were outstanding changes to your files because it will find orphaned recovery files. A dialog box will come up and indicate that a previous session with Source Insight did not end normally. At that point, you have three options.

- Click the **Recover** button. Source Insight will proceed to recover all files previously opened. After recovery, your session should look the same as it was the last time the recovery file was synchronized. All unsaved edits should still be present.

  (or)

- Click the **Continue** button. Source Insight will continue to start as though no recovery were done. You will **not** be able to recover after continuing.

  (or)

- Click the **Quit** button. Source Insight will quit immediately without attempting recovery. If you run Source Insight again, it will still be able to do the recovery if you want to.

When a recovery is performed, Source Insight will resume as though you were in the middle of an editing session. All the previously opened files will still be open and any edits you made will still be there. After the recovery, you can continue to edit normally, or you can quit and save each file that had changes.

Warnings

Verify the results of a recovery before saving.

Source Insight's recovery system is very good, however, no recovery system can be foolproof. There are always windows of vulnerability. If a recovery is needed, it is probably due to unusual circumstances, such as a hardware failure, a power failure, a bug in a program that you spawned from Source Insight, or a bug in Source Insight. *Whatever the reason, it is still a good idea for you to scroll through each recovered file to see that everything looks intact before you actually go ahead and save the files*. If Source Insight's recovery file was damaged or not written entirely, or the failure occurred while the recovery file was being updated, then Source Insight may unwittingly trash your file while it thought it was recovering it. If you find that the recovery didn't work correctly, *do not save the file*. Close the file without saving it. The original, unsaved file will still be intact, although without your last changes saved.

Note: Before a recovery can take place, the recovery system requires that the original files you had open previously must still exist, and be unchanged since Source Insight opened or saved them. If you need to perform a recovery, you should do it right away, before any of the original files are modified.

The Command Line

Source Insight's command line has the following syntax:

```
insight3 [-option] [ [+linenumber file] [+file] [file]
[symbolname] ]
```

Any number of options, files, and symbol names may be given on the command line.

Each option given on the command line must be preceded by a dash (-) or a forward slash (/).

File Arguments

Each "file" argument may be one of the following types:

- A regular file name. Source Insight allows you to omit file extensions. In this case, Source Insight tries to find a match within the current project, as though you typed `<filename>.*`. If more than one file exists with that name, then a list will be shown and you will be able to pick the exact file.
- A symbol name. Since Source Insight treats file names as symbols, each file argument can be more generally thought of as a symbol. Source Insight will open the file where the symbol lives and jump to its location. You may specify any number of symbols. If the symbol name conflicts with file names (without extensions), or with other symbols, then a list will be shown and you will be able to pick the exact symbol instance.
- The name of a workspace file, including the .WK3 extension. Source Insight will open all the files contained in the workspace.
- The name of a configuration file, including the .CF3 extension. Source Insight will load the given configuration file.

Opening Files

On the Source Insight command line, you can specify files following the options. Each file name may be optionally preceded by a plus sign (+) and a line number. If this is done, the file will be displayed with that line number visible.

For example:

```
insight3 +100 file.c
```

This will open FILE.C and position the file so that line 100 is visible.

In addition, each file name may be optionally preceded with a plus sign (+) only. In this case, the file is opened along with the other files in the current workspace.

For example:

```
insight3 +file.c
```

This will open FILE.C along with all the files in the current workspace.

If one or more files are specified on the command line without the plus sign prefix, then the files previously open in the current workspace are not opened. Instead, only the files specified on the command line are opened.

For example:

```
insight3 file.c other.c
```

This will open only FILE.C and OTHER.C. The previously opened files in the current workspace are not opened.

File search path

When a relative file name path is given on the command line, Source Insight first tries to find the file relative to the directory that you started Source Insight in. If it can't find the file and a project is open, it looks for the file in the project's file list.

In other words, you can start Source Insight in any directory and give it a file name, and if the file is in the current project, Source Insight should find it. If there happens to be a file with the same name in the startup directory (or relative to it), then Source Insight will open that file.

Opening Workspaces

In addition to regular text files, you may also specify the name of a workspace file for Source Insight to open.

For example:

```
insight3 myset.wk3
```

This will open all the files in the workspace file myset.wk3.

Any number of workspace file names may also be intermixed with regular file names.

For example:

```
insight3 +100 file.c myset.vw print.c myset2.wk3
```

This will open both the files *and* the workspace files.

Command Line Options

The following options may be included on the Source Insight command line.

Using an Already Running Instance

-i <rest of command line>

This option will direct the rest of the command line to an already running instance of Source Insight, if any. If there are no instances already running, then a new instance is started.

Example:

```
insight3 –i myfile.cpp
```

This will locate an already running instance of Source Insight, and tell it to open `myfile.cpp`.

Running a Command

-c <commandname>

This option will start Source Insight, and run the specified command. The command can be a built-in command, or a defined custom command, or a macro command.

Specifying a Project

-p <projectname>

This option closes the current project, if any, and opens the project given in `projectname`. If the project does not exist, Source Insight will give an error message.

Example:

```
insight3 -p myproj
```

Closing the Current Project

-pc

This option closes the current project if one is open. No other project is opened.

Example:

```
insight3 -pc
```

Using a Temporary Project

-pt <projectname>

This option closes the current project, if any, and opens the project given in projectname. Unlike the -p option, the next time you run Source Insight, the old current project is opened. This is useful if you want to put a Source Insight command in a batch file, but don't want the current project to be changed.

Example:

```
insight3 -pt mail
```

Finding a Symbol

-f <symbol_name>

This option locates the symbol given in symbol_name, opens that file, and positions the insertion point on the symbol. If the symbol can't be found, Source Insight will give an error message. This is unlike specifying a symbol in place of a file name because this explicitly tells Source Insight that you are looking for a parsed symbol, *not* a file.

Example:

```
insight3 -f DoIdle
```

Synchronizing Project Files

-u

This option updates all project files right away when Source Insight starts. It is exactly like using the Synchronize Files command on the Project menu.

Example:

```
insight3 -u
```

Synchronize in Batch Mode

-ub

This option is like -u except that Source Insight quits after the files are updated. This allows you to put a command in a batch file to synchronize Source Insight projects.

Example:

```
insight3 -ub
```

Suppressing the Splash Screen

-s

This option turns off the Source Insight splash screen that normally appears when starting up.

Commands in Source Insight

A *command* is an operation that Source Insight performs when you select a menu item or type a keystroke. For example, the Open command opens a file; the Save command saves a file. Each command has a name, and an action. Commands are resources that can be assigned to menus, keystrokes, and mouse clicks, and those assignments are part of a configuration.

Assign keys to a command with Options > Key Assignments.

Keystrokes and mouse clicks are assigned to Commands. For example, the Ctrl+O keystroke is assigned to the Open command. More than one keystroke may be assigned to a given command. For example, you could assign both Alt-O and Ctrl-O to the Open command. Typing either keystroke would invoke the Open command.

Assign commands to menus with Options > Menu Assignments.

Commands are assigned to menus. For example, the Open command is assigned to the File menu. Its name "Open" appears on the File menu. Selecting that menu item performs the Open command.

Use the Menu Assignments command to customize the contents of the menus. Use the Key Assignments command to customize the keyboard.

Source Insight also allows you to define custom commands, which are useful for launching the compiler from Source Insight. See Also: Custom Commands.

Custom Commands

In addition to the standard commands that are built into Source Insight, you can define custom commands. Custom commands are similar to shell batch files. They execute external shell commands and Windows GUI programs. Source Insight allows custom commands to execute in the background. The output of shell custom commands can be captured into a file for editing, or can be pasted into the current selection.

Custom Commands are shell command that launch from inside Source Insight.

The Options > Custom Command dialog box allows you to define and run custom commands. Once defined, a custom command is like any other command. It can be assigned to a menu or a keystroke can be assigned to it. Custom commands are part of the current configuration.

Custom commands are useful for launching a compilation. By having a custom command that runs the compiler or make program, you can capture the compiler error messages and have the errors parsed and have source links pointing to the erroneous source code automatically.

You can also implement a variety of text filters using custom commands. For example, you could define a Sort custom command that runs a sort filter and pastes the output back over the current selection.

See Also: Custom Commands Keyboard Assignments, Menu Assignments, Save Configuration.

Customizing Source Insight

Source Insight lets you customize many things. The whole set of options is called a *configuration* and configuration files can be saved and loaded.

Your customizations are stored in a configuration file.

Usually, all your customizations are saved in a file called *global.cf3*, stored in the Source Insight program directory, or user data directory. This is a "global" configuration file. You can have a project use its own configuration file with options in the Project Settings dialog box.

Preferences

The Preferences command allows you to set a variety of user options, such as file handling, display options, and language support.

Document Options

The Document Options command allows you to define and change document types. Document types are file types that let you govern Source Insight's behavior depending on the name or extension of each file.

Key Assignments

The Key Assignments command allows you to remap the keyboard in Source Insight. Each command in Source Insight is listed in this dialog box and each command can be given a keystroke or mouse button shortcut.

Menu Assignments

The Menu Assignments command allows you to customize the Source Insight menu bar. Each command in Source Insight is listed in this dialog box and each command can be put on any menu.

Loading and Saving Configurations

You can save a configuration by using the Save Configuration command. You can load a new set of configuration options by using the Load Configuration command. When you load a configuration, it replaces the current configuration stored on disk as well.

You can also save individual parts, such as just the key assignments, with the Save Configuration command.

Project Settings

Project Settings are saved in the project, not the configuration file.

The Project Settings command allows you to set special options for each project. Unlike the other customizations listed previously, the project settings are stored in the project, and not in a configuration file.

Project-Specific Configurations

If you want a project to have its own configuration, use the Project Settings command to specify that. When a project has its own configuration file, then all the user preferences change when you open the project.

Configurations

Customizations are stored in configuration files.

You can customize many aspects of Source Insight, such as the color of the screen and the assignments of keystrokes to commands. This collection of options is called a *configuration*. Configurations usually contain information that rarely changes. You usually set up your configuration once when you install Source Insight, and perhaps occasionally you may modify it to suit your needs or changing tastes.

A configuration contains most of the options specified on the Options menu, and the Preferences dialog box, which include the following things:

- Display Options
- Language Options
- Syntax Formatting Styles
- Syntax Decorations
- Key Assignments
- Menu Assignments
- Toolbars
- Custom Commands
- Document Types and Options
- Page Setup
- File Options
- General Options
- Symbol Lookup Options
- Typing Options

The configuration file is updated when change options.

The configuration in effect while Source Insight is running is called the *current configuration*. When you make customization changes, or load new configurations, the configuration file on disk is updated too.

Configuration Files

When you exit Source Insight, the current configuration is saved in a configuration file. When Source Insight starts, it restores the configuration of the previous session.

If a project is not open, then the name of the current configuration file is global.cf3, and it is stored in your Source Insight program directory (where you told the Setup program to install Source Insight).

If a project is open, the name of the current configuration file is either <projectname>.cf3 or *Global.cf3* depending on the settings in the Project Settings dialog.

Where Are Configuration Files Stored?

Configuration files are normally saved in the Source Insight program directory, or in the user data directories inside it.

Each user that logs in and runs Source Insight gets a user data directory inside the Source Insight program directory. Their configuration files are stored there. Therefore, each user on a particular machine will have their own preferences stored separately.

Tip: It is wise to keep a backup copy of your global configuration file, which will end up containing all your customizations. Once you use the Load Configuration command, or make a change to the customization settings inside Source Insight, the configuration file will be changed automatically.

Tip: It is also a good idea to make a backup copy if you update your Source Insight software. Often, newer builds of Source Insight will be compatible with older configuration files, but not the other way around. If you should wish to revert to an older build of the software, it is best to use an older configuration file.

Loading a Configuration

You can open a configuration file using one of the following methods:

- Use the Load Configuration command and select a new configuration file to be loaded. This command allows loading either the entire configuration contents, or only a specified subset of the configuration, such as just the menu contents.
- Use the Open command and specify the configuration file name, including the .CF3 extension. The entire configuration contents will be loaded, thereby replacing the current configuration.
- Drag and drop a configuration file from Windows Explorer onto the Source Insight window.
- When you open a project that has its own configuration file, then the project's configuration is loaded also.

Saving a Configuration

You can use the Save Configuration command to save the contents of the current configuration to any other configuration file. By saving configuration files, you can create several customized versions of Source Insight, with each one having different menus, keystroke assignments, screen colors, and more. The Save Configuration command also allows you to save a specified subset of the current configuration.

Workspaces

A workspace contains session state that changes from session to session. During an editing session, you may have several files open. Each file has text selected in it. Each file may have bookmarks that you set. This information is part of a workspace. A workspace contains the following things:

- What files you have open
- Selections in each file
- Bookmarks you may have set
- Selection history (where you have been in each file)
- The size and position of each window
- Search and replace strings
- Source links
- Dialog box typing history
- The command recording (see: Start Recording)

When you quit Source Insight, the current workspace is saved in a *workspace file*. When Source Insight starts, it looks in a workspace file to restore the previous session's workspace.

The workspace in effect while Source Insight is running is called the *current workspace*.

If a project is not open, then the name of the current workspace file is *Global.wk3*. If a project is open, the name of the current workspace file is either <projectname>.WK3 or *Global.wk3* depending on the settings in the Project Settings dialog.

When a workspace file is opened, all the files open in that workspace are opened, and all of the above listed workspace elements are restored. After opening a workspace file, the current workspace becomes equivalent to it.

Loading and Saving Workspaces

To open a workspace, use the Open command and specify the workspace file name, including the .WK3 extension, or drag and drop a workspace file from File Manager onto the Source Insight window.

You can use the Save Workspace command to save the contents of the current workspace to any other workspace file.

Managing Tasks With Workspaces

By using multiple workspaces, you can group your tasks and save them individually. Let's say, for example, that you have two different tasks on which you are working. For task 1, you open all the source files for that task and begin working. When you want to stop working on task 1, you use the Save Workspace command to save the current workspace to the file Task1. Next, you close all files, and open the source files to begin working on task 2. When you want to stop working on task 2 and resume working on task 1, you use the Save Workspace command to save the current workspace to Task2, then use the Open command to open the task 1 session, Task1.

Performance Tuning

Depending on the size of your project, and the type of machine you have, you may want to tune Source Insight for better performance. The Preferences dialog box, and the Project Settings dialog box both contain options that affect performance.

Factors That Affect Performance

Source Insight was designed to push some of the limits of functionality for a programming editor. As such, it has features that can stress a system that is too slow or not optimized for it. This section describes factors that you can control which affect Source Insight's performance.

Machine Speed

Pentium II or better is recommended.

A Pentium II machine (or faster) is recommended to take advantage of all the features of Source Insight. If your machine is not quite fast enough, you can turn off many options to improve performance, albeit with less functionality.

In most cases, you can return to version 2.x functionality and speed by selecting the correct options.

Source Insight has many new performance optimizations that allow it to provide more functionality with minimal speed loss.

Project Size

Project size affects performance and memory usage.

The size of a project has a large effect on the performance of certain features in Source Insight. The project size can be measured in number of files, and in number of declared symbols.

Use Project Report to see project size statistics.

To find out how large your project is, use the Project > Project Report command and note the top lines of the report. At the top of the project report, Source Insight prints the number of files, the number of symbols, and the number of symbol index entries in the project. You can also see these same statistics in the Project > Rebuild Project dialog box at the bottom of the dialog box. (You do not have to rebuild the project to see the statistics.)

Source Insight 3.1 handles even more symbol information than version 2.1 did. However, the symbol indexes have grown in size. This may affect very large projects.

Project Index Settings

Project Settings control indexing options for projects.

The Project Settings dialog box contains options for indexing your project. If you enable all the indexing options, and your project is large, you may hurt performance.

The symptoms of too large an index are:

- Disk thrashing while building or rebuilding a large project, with little progress is being made. Note: It is normal for a very large project to require a significant amount of disk access near the end of the rebuilding phase.
- Opening or closing a project takes a long time.
- Synchronizing individual files is slow.
- The Browse Project Symbols dialog box (F7) is slow to come up, accompanied by a lot of disk activity. Note: It is normal for this to pause a second or two the first time you use it.
- Over 1 or 2 million index entries on a 128 megabyte system.

Syllable indexing a large project can be slow and memory intensive

If you have a large project (over 200,000 symbols) you should try turning off the symbol syllable indexing. You can find out how large the database is by selecting Project: Rebuild Project and looking at the statistics on the bottom of the dialog box. Just cancel this dialog box when you are done. If the number of index entries is over 1 million, then things can start to slow down. Adding more memory to your machine will improve performance.

To remove the syllable indexing, run the Project > Project Settings command and turn off **Quick browsing for symbol syllables**. Then use the Project Rebuild command to recreate your project from scratch.

Symbol Memory Usage

Source Insight uses about 16 bytes of RAM for every symbol index entry in a project. A project with all the symbol index options enabled (see Project Settings) will use about 5 or 6 times as many index entries as there are symbol definitions. Therefore, for example, if your project has about 100,000 symbol definitions, then it will have about 500,000 index entries. That will take about 16 x 500,000 = 8,000,000 bytes to hold the index.

Virtual Memory Capacity

Large projects can use a lot of virtual memory

Source Insight uses memory in proportion to the size of your project. If your projects are large, Source Insight will require more memory.

The Win32 programming interface allows programs, like Source Insight, to use much more memory than is physically installed as RAM in your machine. This feature is call *virtual memory*, because the operating system uses the hard disk to emulate RAM, through a process called *paging*.

Source Insight uses memory (and hence, virtual memory) in two ways. First is that Source Insight allocates heap and virtual memory, which may come from the system paging file. This is the smaller type of allocation used. Even so, you will need to make sure your system paging file is large enough to hold approximately 2-5 megabytes for every 100,000 symbols declared in your largest project.

Large project database files are mapped into memory, causing reported memory usage to appear high.

Source Insight also makes heavy use of memory-mapped files. This is a Win32 feature whereby files can be "mapped" into virtual address space, so that a file looks like a block of memory to a program. Source Insight uses memory-mapped files to provide the fastest possible access to source files, and database files.

When Source Insight memory-maps a project database file, it will take up a relatively large amount of virtual address space as soon as the file is opened. As Source Insight accesses different records in the database, the operating system will commit increasingly more physical memory to hold the database file contents. The operating system will always keep some memory in reserve for other programs and the system to use.

The size of a project database is proportional to the number of declared symbols in the project. Therefore, if your project is very large, the amount of memory used by Source Insight can be over 100 megabytes. However, you must realize that most of the memory in use, as reported by the Task Manager, or other performance monitoring tools, represents portions of the symbol database mapped into memory. Source Insight does not require this much physical memory.

Physical Memory Capacity

If your projects are large, Source Insight will require more memory. Having more RAM in your system will improve performance. At least 64 megabytes of RAM is the recommended minimum for Source Insight, and 128 megabytes or more is recommended for sustained use with large projects having over 200,000 symbols.

Operating Systems

Windows 2000/NT is the recommended operating system for Source Insight.

For the best performance with respect to virtual memory, it is recommended that Source Insight run on Windows NT or Windows 2000, or newer. These operating systems utilize memory more intelligently than do Windows 9.x/Me.

Custom Parsing Expressions

Source Insight allows you to augment a language with your own custom parsing regular expressions. You can edit these expressions by right clicking on a source file and selecting Language Properties.

Be careful with the custom parsing expressions you define.

If you have too many expressions, or you are using a regular expression that is inherently slow to match, you may notice that it takes a moment or two to parse your files.

A regular expression can be slow to match if it starts with a pattern that is easily matched, but ends with a pattern that often does not match. For example:

```
[a-z]*(\([a-z]+\))
```

This expression will be slow to match because the pattern starts with `[a-z]*`, which means, "match any zero or more alphabetic characters". Most characters in a file will potentially match.

Speeding Up Program Features

This section contains recommendations for speeding up different aspects of Source Insight.

Speeding Up Syntax Formatting

Source Insight has many interesting and useful display features for formatting source code. Some of those features require a significant amount of processing.

Formatting options affect display speed.

The display code has been sped up considerably to be able to provide more features with acceptable performance. However, if you notice that version 3.1's display speed is a bit slower than version 2.1, keep in mind that a lot more information is being displayed. It is possible to reduce the display functionality to that of version 2.1 and speed it up.

The Preferences: Syntax Formatting dialog box lets you specify how detailed you want the display. Each option has a performance cost. The significant options are listed here, starting with the least costly, to the slowest and most costly:

1. Apply styles for references to members
2. Apply styles for symbol declarations
3. Apply styles for local symbol declarations
4. Apply styles for references to function-local symbols
5. Apply styles for non-function-local symbols
6. Find references using the Project Symbol Path
7. Qualify member references

Formatting "references" is slower than "declarations".

In general, identifying "references" is slower than "declarations". Identifying references to symbols can be slower if the project is very large, because each reference has to be resolved with a symbol lookup.

The symbol lookup engine has been heavily optimized to allow formatting references to symbols at "display speed". A lot of information is cached in the process, so you may notice that the first time you open a file and start scrolling around in it, it may slow down sometimes. Subsequent viewing of the file will speed up.

You can trade-off display richness for speed.

Some people may want the fastest possible display. However, you may be willing to sacrifice a little of that speed in turn for more useful information that increases productivity.

Speeding Up Typing in Browse Dialog Boxes

Typing and filtering symbol lists can become slow if you have a large project, and you have the Project Settings: **Quick browsing for symbol syllables** turned on. Source Insight is trying to perform syllable matching on a large syllable index.

Syllable matching in large projects can be slow.

You can address this two different ways. Either you can turn off syllable indexing in the Project Settings dialog box, or you can change the behavior of the list filtering with the Preferences: Typing dialog box.

The Project Settings options control what is stored and indexed in the symbol database.

The Preferences: Typing options control how the information is used in the lists. You do need the syllable information in the database to use the symbol syllable matching features of the lists. (But, not for file name lists.)

If memory permits, leave Project Settings: **Quick browsing for symbol syllables** enabled, and turn off Preferences: Typing: **Match syllables while typing**. Even with syllable matching while typing disabled, you can still use it by prefixing what you

type with a space. That is, prefixing what you type with a space toggles the syllable matching on and off.

If you encounter a lot of disk thrashing while trying to match any syllables in lists, then your project may be just too big for the available memory.

In Preferences: Typing, turning off both "Match syllables" and "Match members" will speed it up more and return list filtering to version 2.1 functionality.

Speeding Up Building and Synchronizing Projects

A very large project will take a while to build or rebuild. To speed it up, you can try the following:

- In Project Settings, turn off **Quick browsing for symbol syllables**. This index option uses a lot of memory and slows the indexing process.
- In Project Settings, turn off **Quick browsing for member names**. This index option can use a lot of memory also.
- Reduce the number of document types (i.e. file types) that are added to your projects. By default, Source Insight comes with many document types defined, such as Visual Basic files, and ASP files. If you have files like that in your source tree, but you have no need to work on them, then remove them from your project. You can permanently exclude those document types from your projects by running Document Options and un-checking the check box: **Include when adding to projects** – for each type of document you want to exclude from your projects.
- Break your project into smaller projects. If you want to still use the Jump To Definition command between the projects, you can add the "sub-projects" to the project symbol path, which is defined in the Preferences: Symbol Lookups dialog box.

Speeding Up Relation Windows

The Relation Window is a new feature in version Source Insight 3.0. Its purpose is to show relationships between symbols. It works like the Context Window in that it works in the background and automatically shows information about what is selected. For example, if you select a function call, it can show you the call tree starting at that function.

Relation Windows can require a lot of processing.

The Relation Window can require a lot of processing. Some relationships are slower to compute. The relationships fall into three categories, listed here from fastest to slowest:

1. **Contains** – show the contents of the current symbol. For example, show members of a struct.
2. **Uses** – show what other symbols are referred to by the current symbol. For example, show functions that are called by the current function.
3. **References** – show what other symbols refer to the current symbol. For example, show functions that call the current function.

It takes more work to show "references".

For very large projects, the "References" relationship will be by far the slowest to compute. The performance seems very acceptable on a Pentium II machine with a moderate sized project (about 200,000 lines of code).

Limiting the relations to non-reference type relations will speed the Relation Window up.

Locking and refreshing manually usually works well.

It also works well by leaving the Relation Window locked. To lock the Relation Window, click on the lock button in the Relation Window toolbar. You can refresh the Relation Window at any time by using the Refresh Relation Window command, or by clicking on the Refresh button in the Relation Window toolbar.

Speeding Up Auto-Completion

As you type an identifier, the auto-completion window pops up to propose matching identifier names. Every time you type a character, Source Insight considers the symbol data for that file to be "stale". To give you the most accurate auto-completion results, Source Insight would need to reparse your file after each character you type. There is an option that controls this in the Preferences: Symbol Lookups dialog box.

The **Parse locally before lookup** option causes Source Insight to reparse before the auto-completion window appears. On a fast machine, or in a small file, the speed will be acceptable. However, turning this option off will result in a faster response.

Speeding Up Searching Files

There are several ways to search across multiple files in Source Insight. The commands: Search Files, Lookup References, and Search Project all perform multi-file searches.

Lookup References is the fastest type of searching.

The fastest type of search in Source Insight is the Lookup References search. When you search this way, Source Insight looks for references to a single whole-word item. When you search for a single whole-word, Source Insight uses a special index file to make the search fast. It's a good idea to get into the habit of using Lookup References, instead of Search Files, when you can.

Speeding Up Lookup References

The Lookup References command has a few new options that affect its speed.

Context-sensitive options can slow down Lookup References.

The **Smart Reference Matching** option means that the search is context-sensitive; the search results will only contain matches for references to the exact symbol you specify, using the surrounding context. This option slows the process down because each same-string occurrence has to be qualified with a symbol lookup, which requires some parsing on the fly. If you turn this off, it will work like version 2.x of Source Insight.

The **Skip Comments** and **Search Only Comments** options also slow the search down a little, but not as much as the smart reference matching option.

The Smart Rename command also uses these same mechanisms.

Files Created by Source Insight

Source Insight creates the following files and file types on your hard disk.

Files in the Program Directory

The following files are created in the installation directory when Source Insight is installed. The installation directory is the destination directory you specified when you ran the setup program.

File	Description
Insight3.exe	The Source Insight program.
Insight.hlp	The Source Insight Help file.
ReadMe.txt	Text file containing last-minute notes.
Sihook.exe	Utility program used by Source Insight to launch Custom Commands.
Projects.db3	The Project List: contains a list of all projects created on your machine.
Base.*	Project files for the "Base" project.
c.tom	C/C++ token macros.
FileAlias.txt	File name alias file, used to override the document type associated with a given file name.
*.RCO	Crash recovery file, which contains information needed to recover unsaved changes after an abnormal termination.
*.CLF	Custom Language File: this file is created by the exporting a custom language from the Preferences: Languages dialog box.
Backup	Directory containing backup versions of files that are saved with Source Insight.
Clips	Directory containing clips files, which are listed in the Clip Window. You can also copy your own text files to this directory and they will be included automatically in the Clip Window.

Files Created for Each Project

When you create Source Insight projects, the following files are created for each project. In this list, "Name" is the name of a given project.

File	Description
Name.pr	The main project file, which contains a list of the files in the project.
Name.wk3	The project workspace file.
Name.cf3	The project-specific configuration file.
Name.po	Project options.
Name.ps	Symbol definitions database.
Name.pri	Symbol references index.
Name.pfi	Project file index.
Name.imd, Name.imb	Main symbol index.
Name.iad, Name.iab	Auxiliary symbol index for members and syllables.

User Data Directories

Under systems that require a user logon, (such as Windows 2000 or Windows XP), Source Insight creates a user data subdirectory under the Source Insight program directory for each user that runs the program. The user data directory contains the user-specific global configuration file (the user preferences global.cf3 file) and the workspace file for the "no project open" mode.

For example, if John Smith was logged in and ran Source Insight, and Source Insight was stored in c:\Program Files\Insight3, then the following user data directory is created: c:\Program Files\Insight3\John Smith.

The following files are saved in the user data directory.

File	Description
Global.WK3	Global Workspace: the session state used when no project is open.
Global.CF3	Global Configuration: the configuration (user preferences) file used when no project is open, and when a project is open and the Project Settings command specifies "global configuration".

If a configuration file named global.cf3 is saved in the Source Insight program directory, then Source Insight will copy that global.cf3 file to any new users that run Source Insight. Thus, you can use the global.cf3 file stored in the Source Insight program directory as a template configuration for all new users.

Command Reference

This chapter is an alphabetical listing of all the user-level Source Insight commands. Each command is described here.

For more of an overview, see the Source Insight Concepts chapter.

Commands

A Source Insight *command* is an operation that Source Insight performs when you select a menu item or type a keystroke. For example, the Open command opens a file; the Save command saves a file. Each command has a name, and an action. Commands are resources that can be assigned to menus, keystrokes, and mouse clicks, and those assignments are part of a saved configuration.

Use the Menu Assignments command to customize the contents of the menus. Use the Key Assignments command to customize the keyboard.

About Source Insight

The About Source Insight command brings up a window that contains the copyright message and the version number of Source Insight. You should refer to this window to get the version number and build date of Source Insight.

Activate Menu Commands

- Activate Edit Menu
- Activate File Menu
- Activate Help Menu
- Activate Option Menu
- Activate Project Menu
- Activate Search Menu
- Activate View Menu
- Activate Window Menu
- Activate System Menu
- Activate System Doc Menu

The Activate Menu commands activate and "pull down" the menu from the menu bar.

You can also just press and release the Alt key to activate the Source Insight menu bar, then just type a letter to activate the corresponding menu. For example, Alt <release> F activates the File menu.

Source Insight does not force you to follow the Windows standard where Alt+<menu letter> is hardwired to activating that menu. That's because Alt is too good a key to waste on only the menus! Instead, Source Insight allows you to combine Alt with any character and assign it to any command in the Key Assignments command. If you want to make Alt+F activate the File menu, for example, you can just make that key assignment. If you want to make F1 activate the File menu, you can do that too!

Activate Global Symbol List

This command makes the Context Window visible; showing all project symbols in a list, and puts the cursor in the text box at the

top. Once activated, you can type into the text box and the global symbol list will be filtered based on what you type. Pressing Enter or Esc will re-activate your source file window again.

Activate Relation Window

Opens and selects the Relation Window. The input focus is moved to the Relation Window.

Activate Search Results

This command simply activates the Search Results window and brings it to the front, if it is open. This provides a quick way to return to the Search Results.

Activate Symbol Window

This command makes the Symbol Window visible and puts the cursor in the text box at the top. Once activated, you can type into the text box and the symbol list will be filtered based on what you type. Pressing Enter or Esc will re-activate your source file window again.

Add and Remove Project Files

This command lets you to add and remove your source files from the current project.

This is the primary way to add large numbers of files to the project. With this command, you can add and remove single files, groups of files, and whole source directory trees.

What Files Should You Add to a Project?

Add only text files to your projects.

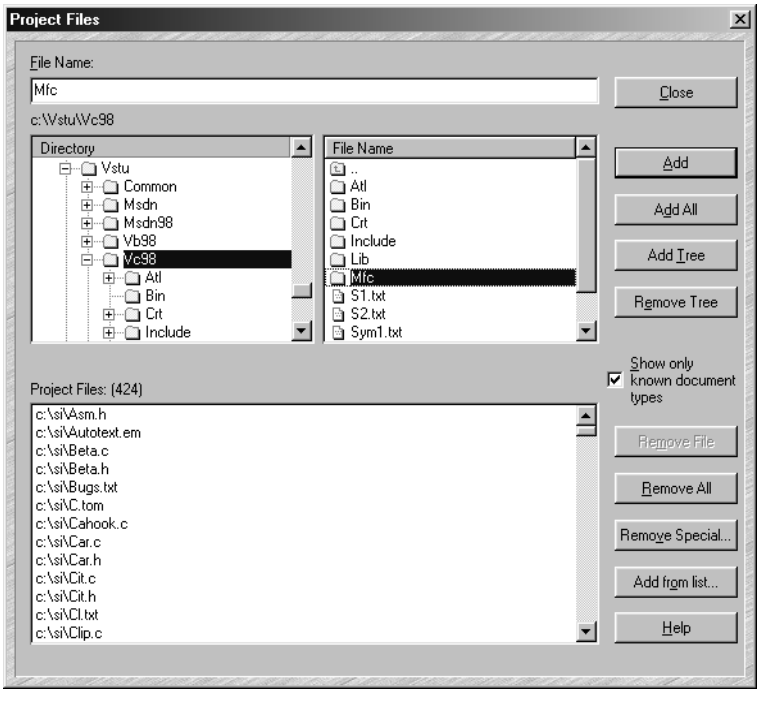
Source Insight projects should consist of program source files and text files only. It doesn't make any sense to add a binary format file to a Source Insight project. For example, adding an EXE file to your project would have no benefit.

The document types that are defined by default correspond to the types of source files you probably want to use with Source Insight. Normally, only those types of files should be added to a project.

Use Document Options to control what types of files are added to projects.

The Document Options dialog box contains the check box: **Include when adding to projects**. You can use this check box to control what file types Source Insight will automatically add to your project, or what file types will be displayed in the list box in the Add and Remove Project Files dialog box.

Add and Remove Project Files Dialog Box



File Name

Type the name of the file you want to add or remove in this text box. The lists will be matched automatically with what you type. You can type a wildcard and press Enter to filter the file list to show only those files that match the wildcard. You can also type a full directory path, or a drive letter followed by a colon to switch the current directory.

Directory List

Contains a directory tree of the current drive. If you select a directory name in this list box, the File Name list will show what is in that directory. The current working directory and wildcard filter, if any, is displayed above the list box.

File Name List

Contains a list of all files in the currently selected directory. If you select a file from this list box, the file name is loaded into the File Name text box. This list box will not display files that are already part of the project.

Project Files List

Lists all the files currently added to the project. You can select files from this list and click the Remove… buttons to remove the files from the project.

Close

Closes the dialog box, keeping the changes you made.

Add

Adds the selected file(s) to the project. If a directory is selected, then the current directory switches to that directory.

Add All

Selects all the items in the File Name list and adds them to the project. If any directories are included, then their contents are added too. Source Insight will prompt you first to see if you want to include the directories.

Add Tree

Click **Add Tree** to add a whole source tree to your project.

When a directory is selected, this adds the whole directory tree to the project. That is, all the directories in the sub tree are scanned for files that match known document types, and they are added to the project.

Remove Tree

When a directory is selected, this removes all files found in that directory tree.

Show only known document types

Only files that belong to known document types are included in the file list. Furthermore, only document types that have the "Include when adding to project" option enabled are included. You can change the known document types with the Document Options command. See Document Options.

If not checked, then all file types are listed in the File Name list.

Remove File

Removes the file(s) selected in the Project Files list.

Remove All

Removes all files from the project. The project will be empty.

Remove Special...

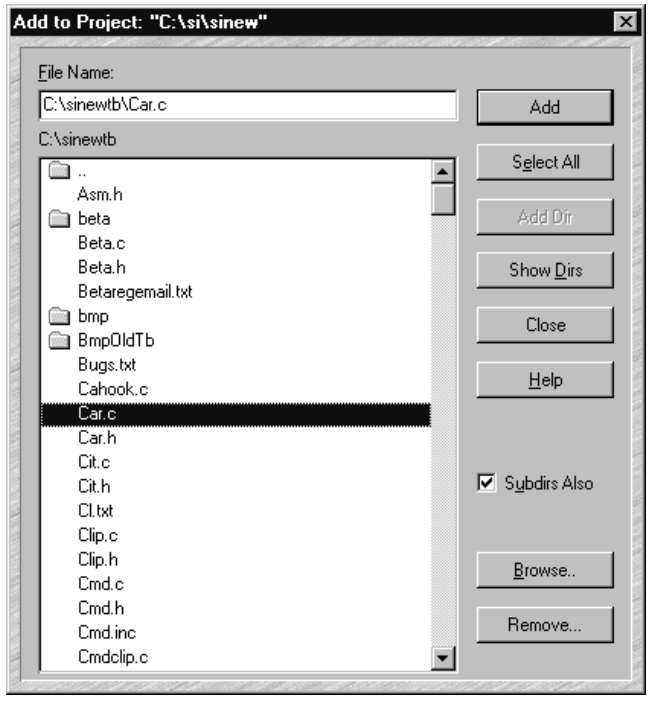
Brings up the Remove File dialog box, which allows you to do special remove operations, such as removing all *.h files.

Add from list...

Brings up the **Add File List** command. This asks you to specify an input text file that contains a list of files and directories to be added to the project.

Add File

The Add File command adds one or more source files to the current project. This command existed in earlier versions of Source Insight, however the **Add and Remove Project Files** command is a replacement, which provides a central dialog box to add *and* remove files from your project.



File Name

The name of the file to be added to the project. You may type a file name or a wildcard pattern and press Enter. If you typed a wildcard, the pattern will be applied to the file list box.

File list box

Contains a list of all files in the current working directory of the current drive that are not already a part of the current project. If you select a file from this list box, the file name is loaded into the File Name text box. The current working directory is displayed above the list box. This list box will not display files that are already part of the project. Also, only files that belong to document types that have the "Include when adding to project" option turned on are included in the list. See Document Options.

Add

Click this button to add the file named in the File Name text box to the project and close the dialog box. If the File Name text box contains wildcard characters, the wildcards will be expanded and displayed in the File list box, and the dialog box will not be closed. If one or more files are selected in the File list box, then all selected files are added to the project.

Select All

Click this button to select all the files contained in the File list box.

Add Dir

Click this button to add a whole directory to the project. If the **Subdirs Also** check box is enabled, then this will recursively add all files in the whole subdirectory tree.

Show Dirs

Click this button to toggle the list box contents between showing file names, and showing subdirectory names.

Subdirs Also

If checked, then Source Insight will recurse through all subdirectories when the Add button is clicked, or when a single directory is selected and Add is clicked.

If not checked, then Source Insight will only add the selected files or the files in the selected directory and will ignore sub-directories.

Browse

Click this button to bring up the standard Windows Open dialog box, which allows you to browse around your disks. If you select a file in this dialog box, its path will be placed into the File Name text box.

Remove

Click this button to switch to the Remove File dialog box.

Add File List

The Add File List command allows you to add file names and directories specified in an input file to the current project.

This is a useful way to let you, or a project administrator, maintain a list of source files and/or source directories, which can be used to build a Source Insight project. This can be done in lieu of adding the files by hand.

Advanced Options

This allows you to selectively disable internal optimizations in Source Insight. This can help to narrow down a possible bug. If you report a bug, you may be asked to make changes in the Advanced Options dialog box to help troubleshoot a problem. Normally, you should have no need to use this feature.

Back Tab

The Back Tab command moves the cursor to the left by one tab stop.

Backspace

The Backspace command backs over the character to the left of the insertion point. If the selection is extended, the text in the selection is deleted instead.

Beginning of Line

The Beginning of Line command moves the insertion point to the beginning of the current line.

Beginning of Selection

The Beginning of Selection command moves the insertion point to the beginning of the current selection if it is extended. If the selection is already an insertion point, then nothing happens.

Blank Line Down

The Blank Line Down command moves the insertion point to the beginning of the next blank line.

Blank Line Up

The Blank Line Up command moves the insertion point to the beginning of the previous blank line.

Block Down

The Block Down command moves the insertion point to the next } brace. This corresponds to the end of the current code block in languages like C/C++ and Java.

Block Up

The Block Up command moves the insertion point to the previous { brace. This corresponds to the beginning of the current code block in languages like C/C++ and Java.

Bookmark

The Bookmark command can be used to add and remove bookmarks, as well as to move to the location of an existing mark. Bookmarks are part of the current workspace.

Name

Type the name of the bookmark here. Source Insight checks to see if the mark you typed matches an existing bookmark name. If it does, then the **Go To** button become the default button. Pressing Enter will position you to that mark. If the bookmark you've just typed does not exist, then the **Set** button becomes the default button. Pressing Enter will create a new bookmark.

Marks list

Displays a list of all the bookmarks currently set. Each item in the list shows the bookmark's name, the file it's in, and the line number it's on. When you select an item in the **Marks list**, the mark name is loaded into the **Name** text box.

Go To

Click this button to jump to the mark that is selected in the **Marks list**.

Set

Click this button to create a new bookmark. The mark's name is taken from the **Name** text box. If the bookmark name is already in the list (i.e. it already exists), then its position will be redefined.

Remove

Click this button to delete the selected marks.

Bottom of File

The Bottom of File command makes an insertion point at the last line in the current file.

Bottom of Window

The Bottom of Window command makes an insertion point at the last visible line in the active window.

Browse Files

The Browse Files command brings up the standard system Open File dialog box so that you can browse the regular file system and open any file. This is unlike the regular Source Insight Open command, which brings up the Project Window, which lists only the files in the current project, regardless of directories.

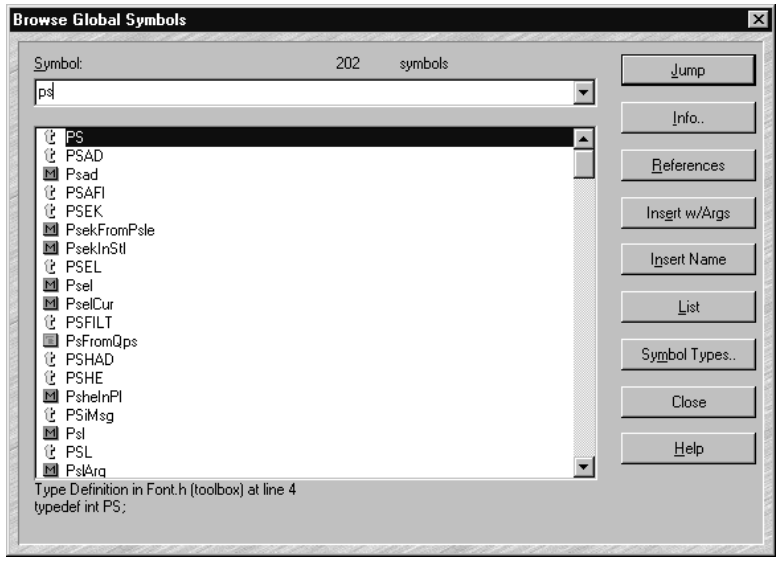
Browse Project Symbols

Lists all the symbols in the current project. From this dialog box, you can

- Find symbols based on parts of their names.
- Look at symbol definitions.
- Jump to symbol definitions.
- Insert a call to a function into your source file.
- Generate a cross-reference listing.

The Browse Project Symbols command automatically selects the first word in the selection before the dialog box comes up. The word is also loaded into the symbol name text box of the dialog box. The word is selected so that you can use the Insert buttons to replace the symbol name.

Browse Global Symbols Dialog box



Symbol

The name of the symbol to be looked up. This text box is automatically loaded with the first word in the current selection when the dialog box comes up. You can type any symbol name into this text box.

Symbol List

This list displays a list of all symbols in the project. If a search pattern was given in the Symbol Name text box, then this is a list of all the symbols that satisfy the search pattern. Below the symbol list, the currently selected symbol's type and file of origin are displayed.

The types of symbols in the list are controlled by the settings accessed with the Symbol Types button.

As you type, Source Insight will display partial matches in the Symbol List. For example, if you type "Pch", then the first item in the list (in sorted order) that starts with "Pch" is selected in the list. The match is not case sensitive and leading underscores are ignored.

If you have symbol syllable matching enabled (in Preferences: Typing) then the Symbol List will also show matches on syllables, which you may type in any order. For example, if you type "cre win" (note the space between items), the Symbol List will show all symbols that have "Cre" and "Win" somewhere in the name.

To temporarily toggle syllable matching on and off, prefix your entry with a space character.

You can search for symbols using regular expressions, by prefixing your pattern with a question mark (?).

You can specify a regular expressions style search pattern to search for symbols by typing a question mark (?) and then the search pattern, and click the Jump button. All the symbols that match the pattern are placed in the Symbol List. For example, to find all symbols beginning with "Delete" and containing "Foo", you could type "?^Delete.*Foo".

Jump

Click this to jump to the definition of the currently selected symbol. If an item is selected in the Symbol List, then that is the current symbol. Otherwise, the symbol typed in the Symbol Name text box is used.

If the symbol name text box starts with a question mark (?), then Source Insight will replace the list with all the symbols that match the search pattern that follows the question mark; the dialog box will remain open.

Info

Click this button to run the Symbol Info command on the selected symbol.

References

Click this button to search for references to the selected symbol.

Insert w/Args

Click this button to replace the current selection with the name of the symbol, followed by the parameters as they appear in the symbol definition, if the symbol is a function.

Insert Name

Click this button to replace the current selection with the name of the symbol.

List

Click this button to create a cross-reference list of the symbols currently listed in the Symbol List. A new file is created and named Symlist.txt. Each line of the file contains a symbol name, and the file and line number where it's defined.

Symbol Types

This button is used to specify what types of symbols will be included in the symbol list and what types of symbols will be

searched for when using a regular expression in the symbol name text box.

Making a Cross Reference

You can have Source Insight create an output file that contains a list of symbol names.

To create a symbol cross reference list:

1. Run the Browse Project Symbols command.
2. Click the List button. A new file named Symlist.txt will be created containing a list of all the symbols displayed in the Symbol List, along with the file and line number where the symbol is found.

To create a partial symbol cross reference list:

1. Run the Browse Project Symbols command.
2. Click the Symbol Types button to specify the desired types of symbols to appear in the list.
3. Type a search expression in the Symbol Name text box. You must begin the pattern with a ? character to indicate that it is a pattern. For example, "?Word" searches for all symbols containing the substring "Word".
4. Click the Jump button to replace the list contents with all the symbols that match the pattern. When the search is done, all the matching symbols will appear in the Symbol List box.
5. Click the List button to create the symbol list.

Inserting a Function Call

You can use the Insert buttons to make entering function calls in your source files easier.

Example 1: Simple function call insertion:

1. Select the place in your file where you want to insert the function call.
2. Run the Browse Project Symbols command. Locate the function you want in the Symbol List.
3. Click the Insert w/Args button. The function definition, including parameters will be inserted into the text of your file.
4. Fix up the arguments in the function call to be appropriate.

Example 2: Inserting a function call:

Let's say you want to call a function but can't remember its name. You know it has "Insert" in its name. This example assumes you **have not** enabled the symbol syllable matching when you created the project.

1. Select the place in your file where you want to insert the function call.
2. Run the Browse Project Symbols command.
3. Turn on the Function check box exclusively in the Criteria group.
4. Type "?Insert" in the File Name text box and click OK. Source Insight will search the symbol list and fill the Symbol List with the search results.
5. You see the function you want in the Symbol List. Select it and click Insert w/Args. The function definition is inserted into your file.
6. Fix up the arguments in the function call to be appropriate.

Example 3: Inserting a function call:

Let's say you want to call a function but can't remember its name. You know it has "Insert" and "Char" in its name. This example assumes you have enabled the symbol syllable matching when you created the project.

1. Select the place in your file where you want to insert the function call.
2. Run the Browse Project Symbols command.
3. Type "Insert Char" in the File Name text box and wait a moment. Note the space between the two words. Source Insight will use the syllables you typed to filter the Symbol

List to show all symbols with Insert and Char in the name. This technique only works if each word you type starts with an uppercase letter.

4. You see the function you want in the Symbol List. Select it and click Insert w/Args. The function definition is inserted into your file.
5. Fix up the arguments in the function call to be appropriate.

Example 4: Inserting function arguments:

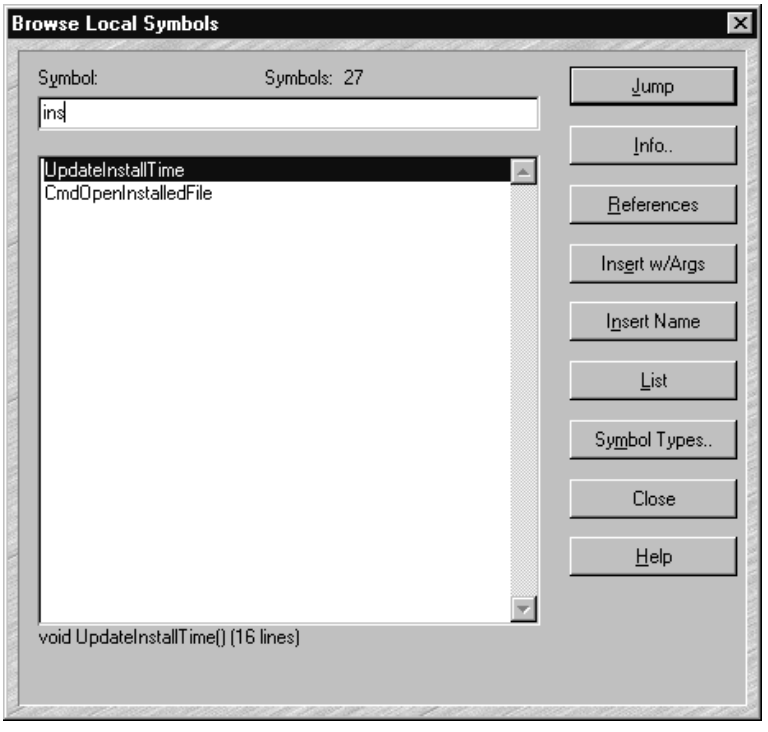
You've just typed the name of a function in your file, but can't remember what parameters it takes.

1. Run the Browse Project Symbols command. Before the dialog box comes up, the first word in the selection is selected, so the function name you've just typed is selected.
2. Locate the function.
3. Click the Insert w/Args button. The current selection (the function name you typed) is replaced with the function name and parameters.
4. Fix up the arguments in the function call to be appropriate.

Browse Local File Symbols

The Browse Local File Symbols command lists all the symbols in the current file that are at the file scope. From this dialog box, you can look at the symbol definition, jump to the symbol, or insert a copy of the symbol definition into the current selection.

The Browse Local File Symbols command automatically selects the first word in the selection before the dialog box comes up. The word is selected so that you can use the Insert buttons to replace the symbol name.



Symbol

The name of the symbol to be looked up. This text box is automatically loaded with the first word in the current selection when the dialog box comes up. You can type any symbol name into this text box.

Symbol List

Displays a list of all symbols in the project. If a search pattern was given in the Symbol Name text box, then this is a list of all the symbols that satisfy the search pattern. Below the symbol list, the currently selected symbol's type and file of origin are displayed.

The types of symbols shown in the list are controlled by the settings accessed with the Symbol Types button.

As you type, Source Insight will select the symbol in the Symbol List that starts with what you are typing. For example, if you type "Pch", then the first item in the list (in sorted order) that starts with "Pch" is selected in the list. The match is not case sensitive and leading underscores are ignored.

You can specify a regular expression style search pattern to search for symbols by typing a question mark (?) and then the search pattern, and click the Jump button. All the symbols that match the

pattern are placed in the Symbol List. For example, to find all symbols beginning with "Delete" and containing "Foo", you could type "?^Delete.*Foo".

You can also specify a file name without drive or directory path. Since Source Insight treats file names as symbols, it will "lookup" the file name by following the normal symbol search rules.

Jump

Click Jump to jump to the definition of the currently selected symbol. If an item is selected in the Symbol List, then that is the selected symbol. Otherwise, the symbol typed in the Symbol Name text box is used.

If the symbol name text box starts with a question mark (?), then Source Insight will perform a search (see *Symbol Name*), and the dialog box will remain up.

Info

Click this button to run the Symbol Info command on the selected symbol.

References

Click this button to search for references to the selected symbol.

Insert w/Args

Click this button to replace the current selection with the name of the symbol followed by the parameters as they appear in the symbol definition.

Insert Name

Click this button to replace the current selection with the name of the symbol.

List

Click this button to create a cross-reference list of the symbols currently listed in the Symbol List. A new file is created and named SYMLIST.TXT. Each line of the file contains a symbol name and the file and line number where it's defined.

Symbol Types

This button is used to specify what types of symbols will be included in the symbol list and what types of symbols will be searched for when using a regular expression in the symbol name text box.

Cascade Windows

The Cascade Windows command rearranges the windows by cascading them down the screen.

Checkpoint

Saves the current file to disk and erases its change history and undo history. You can think of this as a "clean" save operation. It has the same effect as saving the file, closing it, and opening it again. After using Checkpoint, you will not be able to undo any prior changes.

In versions of Source Insight earlier than version 3.1, this command was simply known as Save, because earlier versions did not preserve undo and change history after saving a file.

Checkpoint All

Performs the Checkpoint command on all open files. This saves all open files to disk and erases their undo and change histories. After using Checkpoint All, you will not be able to undo any prior changes in your files.

Clear Highlights

Removes all word highlighting in all source windows. Highlighting is applied by using the Highlight Word command.

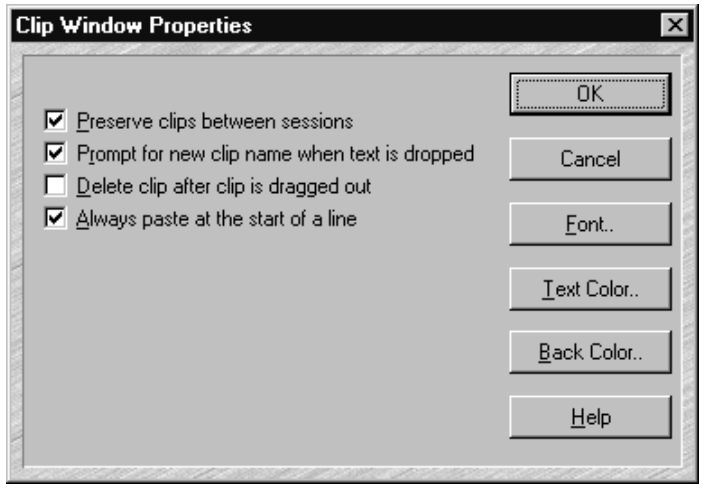
Clip Properties

(On Clip Window tool bar and right-click menu)

The Clip Properties command allows you to edit the name of the clip.

Clip Window Properties

This command brings up the Clip Window Properties dialog box and allows you to set options for the Clip Window.



Preserve clips between sessions

If checked, then clips are automatically saved to the Clips directory (in the Source Insight program directory) and will be reloaded the next time you run Source Insight. If not checked, then clips are thrown away when Source Insight exits.

Prompt for new clip name when text is dropped

If checked, then Source Insight prompts you for the name of a clip whenever you drop text on the Clip Window. If not checked, then Source Insight will generate a simple name for the clip automatically.

Delete clip after clip is dragged out

If checked, then when you drag a clip out of the Clip Window, the clip will be deleted from the Clip Window. If not checked, then the clip will be retained.

Always paste at the start of the line

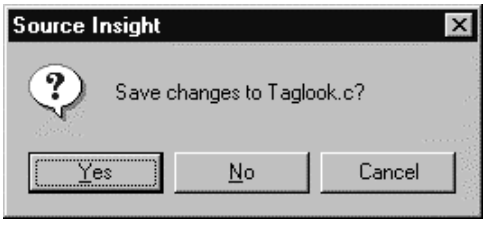
If checked, then the clip will be pasted at the beginning of the current line, instead of exactly where the cursor is. This does not apply when you drag a clip out of the window to a particular spot.

Font, Text Color, Back Color

Lets you pick the display options for the Clip Window.

Close

The Close command closes the current file. If the file has been edited, but not saved, then Source Insight will ask you if you want to save the changes before closing the file by using a dialog box with the following buttons.



Yes

Click to save and close the file.

No

Click to close the file without saving it. Changes you've made will not be saved.

Cancel

Click to cancel the Close command. The file will remain open, and it will not be saved.

Close All

The Close All command performs a Close command on each open file. For any files that you have changed but not saved, Source Insight will ask if you want to save them.

If you have any captured custom command windows open and the custom command is still running, those windows are not closed.

Close Project

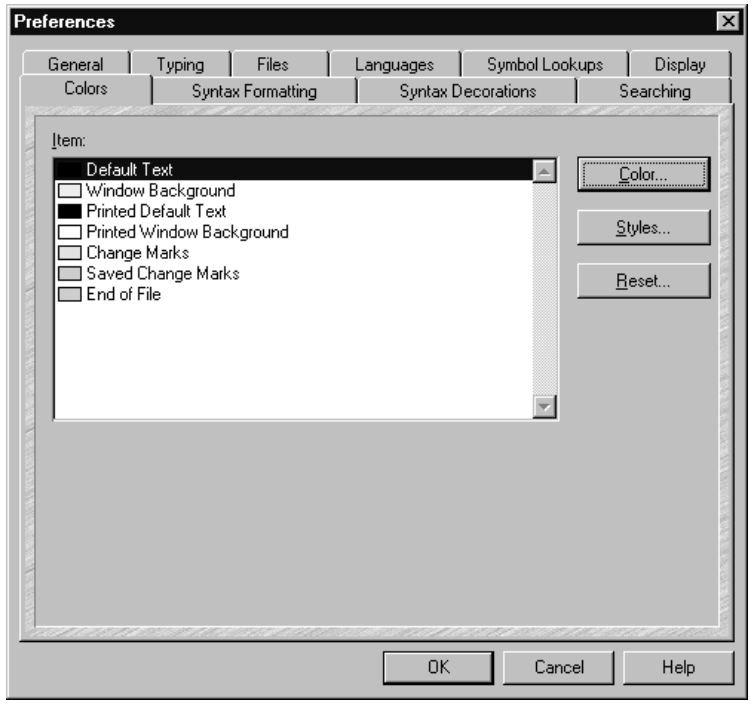
The Close Project command closes the current project. When the project is closed, all open files are also closed. Source Insight does this by performing the Close command on each open file. The workspace and configuration files for the project are also saved.

Close Window

The Close Window command closes the current window.

Color Options

Activates the Preferences: Colors dialog box, which allows you to specify the colors of user interface items.



Item list

Lists the display items that can be colorized. The list contains the following items:

Item	Description
Default Text	Plain text that has no other style applied.
Window Background	The color of the screen window background.
Printed Default Text	The printed color of plain text.
Printed Window Background	The printed background color.
Change Marks	The color of the change marks that appear in the left margin alongside lines that have been edited, but not saved.
Saved Change Marks	The color of the change marks that appear in the left margin alongside lines that have been edited, after the changes have been saved to disk.
End of File	The color of the area that appears below the end of files.

Color

Click this button to select a new color.

Styles

Edits the style properties.

Reset

Resets the color options to the initial defaults.

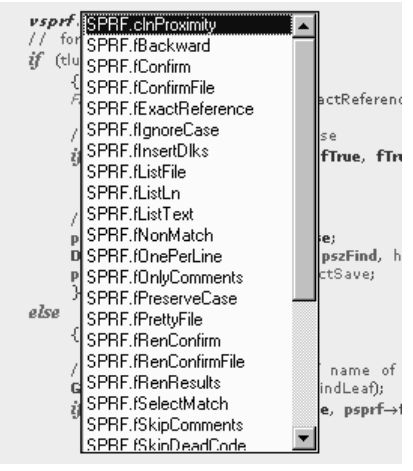
Command Shell

The Command Shell command is a Custom Command that launches a shell command box from Source Insight.

Complete Symbol

This command completes the entry of a symbol name when you are typing, thereby saving you from typing the whole name.

If auto completion is enabled, then the Complete Symbol command invokes the auto-completion function. The popup auto-completion window will appear.



The Auto-Completion window appearing after typing

If auto completion is not enabled and the Context Window is visible, then as you type a symbol name, the Context Window begins showing you the names of symbols that partially match what you have typed so far.

The Complete Symbol command replaces the whole word you are typing with the complete name of the symbol.

For example, let's say that you have a function called `InitWindowState`. As you type in the letters: "`InitWi`", the auto completion window, or the Context Window, narrows what you've typed down to a unique function (`InitWindowState`). The Complete Symbol command will replace what you've just typed with the whole name "`InitWindowState`".

The auto completion settings affect how the completion function works. You have the option of inserting function call parameters when a function name is inserted. You can change the auto-completion options using the Preferences command.

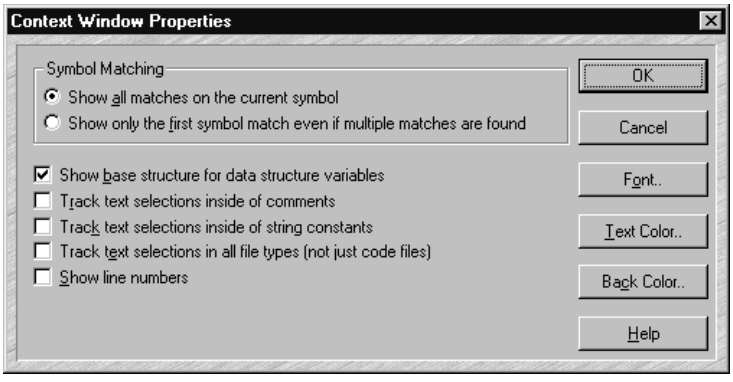
Context Window

This command toggles the Context Window visibility on and off. The Activate Context Window command also makes the Context Window visible, and it changes focus to the Context Window.

Context Window Properties

This command allows you to specify properties for the Context Window. The Context Window tracks what you select and type in

text windows, as well as what files you select in the Project Window, Relation Window, and Clip Window.


Context Window Properties
Symbol Matching
Show all matches on the current symbol
Show only the first symbol match even if multiple matches are found
Show base structure for data structure variables
Track text selections inside of comments
Track text selections inside of string constants
Track text selections in all file types (not just code files)
Show line numbers
OK
Cancel
Font..
Text Color..
Back Color..
Help

Show all matches on the current symbol

Turn this **on** to have the Context Window show a complete list of all symbol matches in its list. If you have multiple definitions with the same name, then they will all appear in the list.

Show only the first symbol match

Turn this **on** to show only first matching symbol's definition. If you often have symbols that are defined in more than one place but are essentially the same thing, then you might want to turn on this radio button.

Show base types for data structure variables

Check this to have Source Insight decode variable declarations and try to locate base structure-type definitions (i.e. structs, unions, classes, etc.).

For example:

```
struct S { int x; };
struct S *psvar;

psvar->...
```

If you select inside of `psvar`, then Source Insight will look at the declaration of `psvar`, see that it is a pointer to `struct S`, and then show you the declaration of `struct S`.

Uncheck this option to simply show the declaration of the each variable, without walking up the declaration hierarchy.

Track text selections inside of comments

Turn this **on** to have the Context Window look up symbols when the cursor is inside of comments.

Track text selections inside of string constants

Turn this **on** to have the Context Window look up symbols when the cursor is inside of quoted string constants.

Track text selections in all file types

Turn this **on** to have the Context Window look up symbols when the cursor is inside any type of file, not just source code files.

Show line numbers

Makes line numbers visible in the Context Window.

Font

Specifies the font used to display symbol lists and source code in the Context Window. If the Context Window is currently showing a list, then this specifies the list font. If the Context Window is currently showing a source file, then this specifies the source code font.

Text Color

Specifies the text color of list items in the Context Window. The colors of source code are determined by the Syntax Formatting options in the Preferences dialog box.

Back Color

Specifies the background color of list items in the Context Window. The colors of source code are determined by the Syntax Formatting options in the Preferences dialog box.

Copy

The Copy command copies the contents of the current selection to the clipboard. Once in the clipboard, it can be pasted to other locations using the Paste command. This command is only allowed if the current selection is extended.

Copy Line

The Copy Line command extends the current selection to include whole lines and copies that selection to the clipboard. Each use of Copy Line extends the selection down one more line.

Copy Line Right

The Copy Line Right command copies the text from the insertion point to the end of the current line into the clipboard.

Copy List

This command appears on the right-click menu when you click on a list. It copies the contents of the list to the Clipboard. This lets you make a copy of any list, or paste any list into a file and print it.

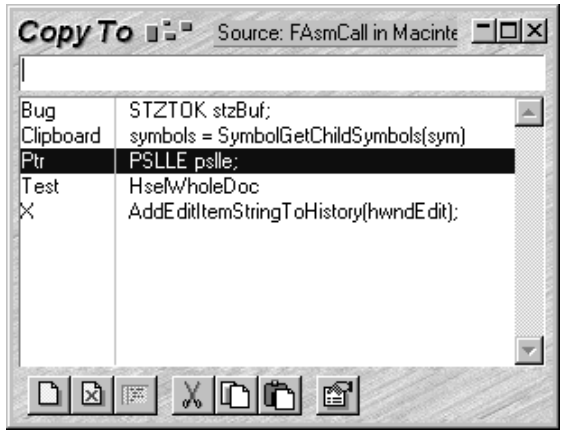
Copy Symbol

This command appears on the Symbol Window right-click menu.

The Copy Symbol command copies the selected symbol, along with its definition body, to the Clipboard. For example, if you click on a function name and select Copy Symbol, then the whole function is copies to the Clipboard.

Copy To Clip

The Copy To Clip command copies the contents of the current selection to a clip buffer that you name. The Clip Window is activated when you use this command so that you can specify the destination clip name.



After either selecting a clip item in the list, or typing a clip name, press the Enter key to complete the copy operation.

Create Key List

The Create Key List command generates a new file named Keylist.txt containing a list of all commands and the keystrokes assigned to each command. The Keylist.txt file is just a regular, unsaved file. You can edit the file normally. This is useful for creating your own keyboard quick reference guide.

Create Command List

This generates a new file named "Command List", which contains a list of all commands and their descriptions. The output file is just a regular, unsaved file. You can edit the file normally. This is useful for creating your own command quick reference guide.

Cursor Down

The Cursor Down command moves the insertion point down by one line.

Cursor Left

The Cursor Left command moves the insertion point left by one character.

Cursor Right

The Cursor Right command moves the insertion point right by one character.

Cursor Up

The Cursor Up command moves the insertion point up by one line.

Custom Commands

Custom commands are similar to command shell batch files. They allow Source Insight to spawn any command line driven tool and to capture its output. Custom commands can also execute Windows programs.

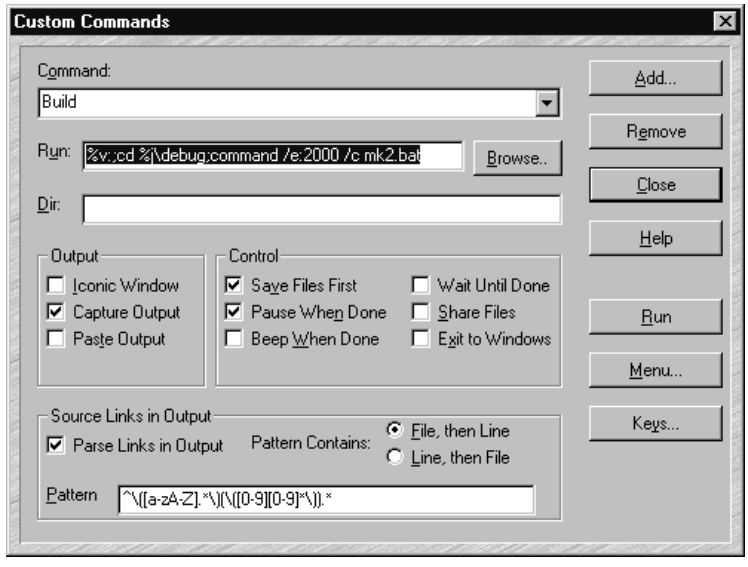
Capturing Command Output

Source Insight allows custom commands to execute, and then return to Source Insight. The output of shell custom commands can be captured into a file for editing, or can be pasted into the current selection. Custom commands are stored as part of the current configuration.

Custom commands can be used to spawn compilers, source control programs, and file filters, such as “sort”.

Note: A shortcut for editing a custom command is to hold down the Ctrl+Alt+Shift key while selecting the command. The Custom Commands dialog box will appear for that command.

Custom Command Dialog box



Command

Displays the name of the currently selected command. This pull-down list contains a list of all the custom commands defined.

Run

This is the command line script to be executed when the custom command is invoked. The **Run** text box can contain special meta-characters. See "The 'Run' Field Format" on page 204.

Dir

The working directory used when executing the script specified in the Run text box. Source Insight sets the current working directory to this location before running the command. If left blank, then Source Insight sets the current working directory to the project root directory.

Ouput Group

This group of options control what happens to the output of the command.

Iconic Window

If checked, the spawned program will be put into a minimized window. If not checked, then the program will launch normally.

Capture Output

If checked, the standard output of the command will be captured and will appear in a new command output window when the command completes. The command output window's title will be the name of the custom command. If not checked, the standard output will not be captured.

Paste Output

If checked, the standard output of the command is pasted to the current selection.

Control group

This group of options specifies what Source Insight does before and after the command runs.

Save Files First

If checked, Source Insight will perform a Save All command prior to executing the command. The Save All command will prompt you for each file that has been edited to see if you want to save the file.

If not checked, the command will be executed without saving any changed files. *The unsaved changes will be retained* when the command completes and control returns to Source Insight. If the command should cause a crash, Source Insight will be able to perform a recovery and all changes will be intact. See Also: Recovering from Crashes.

Pause When Done

If checked, Source Insight will display this message in a DOS box when the command completes:

```
Press any key to return to Source Insight...
```

If not checked, the DOS box will terminate after the command completes.

Beep When Done

If checked, Source Insight will beep when the command terminates.

Wait Until Done

If checked, then Source Insight will suspend itself until the command finishes.

If not checked, then Source Insight will run the command and continue. You will be able to switch back to the Source Insight window and continue working while the command runs in the background.

Exit Source Insight

Source Insight will exit after launching the program.

Source Links in Output

These options specify how the output is to be treated after the command finishes. You may tell Source Insight to parse through the captured output to find specific warning or error messages.

Parse Source Links

The command output will be searched for source link patterns. The patterns typically will match warning and error messages. If a pattern match is found, Source Insight inserts a source link at that line. The source link is used to link the warning or error message to its target source line. If **Parse Source Links** is enabled, you must have a valid search expression in the **Pattern** text box.

Pattern contains

File, then Line and ***Line, then File*****.** This indicates the order of the groups in the pattern expression.

Select **File, then Line** if the first group in the pattern expression is the file name and the second group is the line number. With this setting, the second group, (i.e. the line number), is optional.

Select **Line, then File** if the first group in the pattern expression is the line number and the second group is the file name.

Pattern

Contains the regular expression used to search the command output for file names and line numbers. This is ignored if the **Parse Source Links** option is disabled. If the option is enabled, then this text box must contain a valid regular expression that contains "groups" for the file name and the line number. See "Regular Expressions" on page 125.

Define

Click this button to define the command named in the Name text box. If the command already exists, it is redefined.

Remove

Click this button to delete the command.

Run

Click this button to define and execute the command.

Cancel

Click this button to cancel the dialog box. Any definitions made in the dialog box will be retained.

Menu...

Click this button to define the current command and jump to the Menu Assignments dialog.

Keys...

Click this button to define the current command and jump to the Key Assignments dialog.

The 'Run' Field Format

The **Run** text box contains the command line to execute when the custom command is invoked. The **Run** text box can contain more than one command. Each command should be separated by a semi-colon. For example,

```
cat make.log;echotime
```

This string causes "`cat make.log`" to execute, followed by "`echotime`".

Running the Command Shell

If you want to run a shell command, such as "type", or "dir", or you want to run a batch file, then you have to run command.com first. For example,

```
command /c mybat.bat  or
command /c type foo.txt
```

If you are using Windows NT or Windows 2000, you should use cmd.exe instead of command.com.

If the **Run** string contains more than one command, separated by semi-colons, you don't need to run command.com because Source Insight creates a batch file from the run string commands and runs command.com automatically in that case. For example,

```
cat readme.txt;dir
```

This works fine because it is already running in a batch file inside a shell.

You may find that the shell you spawned by command.com does not have enough environment space. If that happens, use the /e switch with command.com. For example,

```
command /e:1024 /c mybat.bat
```

This allocates 1K bytes of environment space to the new sub shell spawned by command.com.

Meta-Characters

The **Run** text box can contain meta-characters that cause the following items to be substituted in the string.

Character	Expands to	Example
%f	full path name of the current file *	c:\myproj\file.c
%r	path name of current file relative to the project root directory *	file.c
%n	leaf name of the current file *	file.c
%d	directory path of the current file	c:\myproj
%h	directory path of current file without the drive letter	\myproj
%b	leaf name of current file w/o extension *	file
%e	extension of the current file	c
%c	drive letter of the current file	c:
%p	the current project name	c:\myproj\myproj
%j	the root directory of the current project	c:\myproj
%v	the drive letter of the current project's root directory	c:
%o	leaf name of the project without path	myproj
%l	the current line number	any number
%w	first word in the selection, or the word under the cursor	any word
%s	name of a temp file where the current selection is saved	d:\tmp\vt0004.
%a	the current date	05-12-89
%t	the current time	08:23
%1 - %9	user is prompted for arguments	any strings

You can also postfix any of the above characters marked with * with either of the following modifier characters.

Character	Expands to	Example
%o	for all open files	`%f%o`
%m	for all modified files	`%f%m`

ShellExecute Commands

ShellExecute lets you invoke Windows Shell commands.

Custom Commands support the "ShellExecute" function, which lets you tell the Windows shell to perform an operation on a specified file. The nice thing about ShellExecute is that you don't need to know what specific application is registered to handle a particular type of file. For technical background information, see the "ShellExecute" function in the Windows Shell API documentation.

To use this feature, the **Run** string in the custom command needs to start with "ShellExecute". The format should be:

```
ShellExecute <verb> <filespec> <optional parameters>
```

For example, to browse a website:

```
ShellExecute open http://www.somedomain.com
```

The `verb` is a single word, which can be one of the following:

- **`edit`** - Opens an editor for the file.
- **`explore`** - The function explores the folder specified.
- **`open`** - The function opens the file specified. The file can be an executable file or a document file. It can also be a folder.
- **`print`** - The function prints the document file specified. If `filespec` is not a document file, the function will fail.
- **`properties`** - Displays the file or folder's properties.
- **`find`** - Launches the Find Files application found on the Start menu.
- `""` (empty string) to skip this parameter to ShellExecute.

The `filespec` parameter can be any valid path. Use double quotes around complex path names with embedded spaces. You can also use a meta-character, such as `%f` (for the current file). It can also be the name of an executable file.

The `optional parameters` list is anything to the right of the `filespec`. It specifies the parameters to be passed to the application that ultimately runs. The format is determined by the

verb that is to be invoked, and the application that runs. You can use custom command meta-characters here as well.

The working directory text box of the custom command is applied before the ShellExecute is invoked. However, output cannot be captured or parsed when using ShellExecute.

ShellExecute Examples

Here are some useful examples showing how to use ShellExecute.

Action	Custom Command Run String
To browse to a web site:	`ShellExecute open http://www.somewebsite.com`
To explore your Windows 2000 documents file folder:	`ShellExecute explore "C:\Documents and Settings"`
To explore your Windows 98 documents file folder:	`ShellExecute explore "C:\My Documents"`
To launch Internet Explorer:	`ShellExecute "" iexplore`
To preview a file in Internet Explorer:	`ShellExecute "" iexplore %f`
To search for files in the current project folder:	`ShellExecute find %j`

Running Custom Commands in the Background

When Source Insight spawns a Custom Command shell program, it actually runs a program called Sihook3.exe. Sihook3.exe in turn spawns the command and performs the output capturing. You can run a custom command and click back on the Source Insight window to continue editing with Source Insight while the custom command runs in the background.

Creating a Compile command

You can spawn a compiler from Source Insight, using a custom command, and have the output captured and parsed for error messages. Then you can use Go To First Link and Go To Next Link to view each error in your source files.

If you are using the Microsoft ® C++ compiler, to create a simple Compile command:

1. Run the **Custom Command** command.
2. Type "Compile File" in the Name text box.
3. In the **Run** text box, typing "cl %f". This invokes the compiler on the current file. You could also invoke a "make" program or a batch file here instead. If you use a batch file, you must run command.com first. For example, "command /c mybatch.bat".
4. Turn on the **Parse Source Links** option. The default parse pattern is setup to parse the compiler error messages from the compiler output.
5. Turn on the **Save Files First** option so that your file is saved before running the compiler.
6. Click the **Define** button to save the new command. Alternatively, you can click the **Menu** or **Keys** buttons to define the new command and run the **Menu Assignments** or **Keyboard Assignments** commands, which will allow you to put the command on a menu or assign a key to it.

The **Parse Source Links** option causes Source Insight to search the compiler output and setup source links for each error message. In this case, the "link sources" are each error message in the compiler output file. The "link target" for each link is the file and line number given in each error message.

Viewing Compiler Errors

To view source lines with errors:

1. Run the **Compile File** custom command, which is defined as described above.
2. Assuming there are errors, when the compiler finishes the error messages will be in the "Compile File" window. Source Insight will automatically setup the source links and run the **Go To First Link** command. The first error message and the erroneous source line will be selected and made visible.
3. Run the **Go To Next Link** command. The next error message in the "Compile File" window is selected, and the target of that link is shown, as was the first error.
4. Continue to use the **Go To Next Link** command until all the links (error messages) have been visited. If there are no

more links, then Source Insight beeps and the message, "No links." will appear in the status bar.

Cut

The Cut command copies the contents of the current selection to the clipboard, and deletes the selection. Note that although the Cut command deleted the current selection, you still have that text saved in the clipboard. You could reverse the deletion by following the Cut command with a Paste command.

Cut Line

The Cut Line command copies the current line to the clipboard and deletes the line. The cursor can be anywhere on the line when you use this command.

Cut Line Left

The Cut Line Left command cuts all the characters to the left of the insertion point on the current line.

Cut Line Right

The Cut Line Right command cuts all the characters to the right of the insertion point on the current line.

Cut Selection or Paste

The Cut Selection or Paste command performs a Cut command if the current selection is extended, or it performs a Paste command if the selection is an insertion point.

If you assign this to a key or mouse button, this command makes moving text around easy because you only have to press a single key or mouse button.

To use this command:

1. Select the text you want to move by clicking and dragging with the left mouse button.
2. Click the mouse button or press the key to cut the text.
3. Point and click the mouse button to select the new location.
4. Click the mouse button or key to paste the text.

Cut Symbol

(On the Symbol Window right-click menu)

The Cut Symbol command cuts the selected symbol.

Cut To Clip

The Cut To Clip command copies the contents of the current selection to a clip buffer that you name, and then deletes the text. The Clip Window is activated when you use this command so that you can specify the destination clip name.

Cut Word

The Cut Word command cuts the word to the right, starting at the insertion point.

Cut Word Left

The Cut Word Left command cuts the word to the left, starting at the insertion point.

Delete

The Delete command deletes any file on disk, including the currently open file.

Delete All Clips

This command deletes all user clips from the Clip Window. The Clipboard is a special clip that cannot be deleted.

Delete Character

The Delete Character command deletes the character at the insertion point. If the current selection is extended, it deletes the whole selection.

Delete Clip

(On Clip Window tool bar and right-click menu)

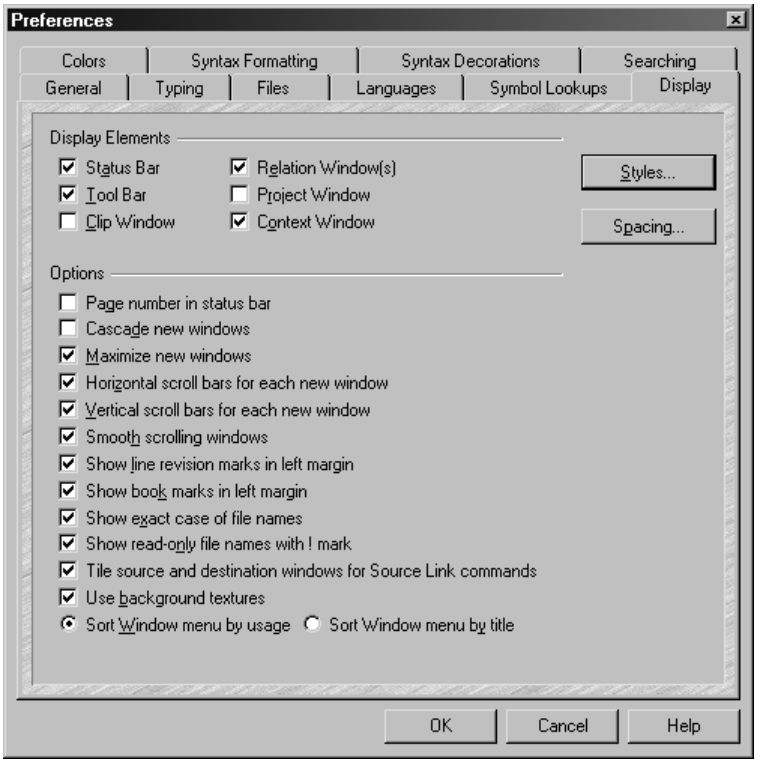
The Delete Clip command deletes a clip file from the Clip Window and from disk.

Delete File

The Delete File command deletes a file from the disk and removes it from the current project if it was part of the project. If you specify the current file, then that file is closed and deleted.

Display Options

This command brings up the Display page of the Preferences dialog box. These options are part of the current configuration.



Display Elements

The Display Elements group is used to turn on and off elements of the user interface.

Status Bar

Turns on and off the status bar at the bottom of the Source Insight application window.

Tool Bar

Turns on and off the main toolbar at the top of the Source Insight application window.

Clip Window

Turns on and off the Clip Window.

Project Window

Turns on and off the Project Window.

Context Window

Turns on and off the Context Window.

Relation Window(s)

Turns on and off all Relation Windows.

Options Group

The items in the Options Group control general options for different display elements in the program.

Page Number in status bar

If checked, then the current page number that contains the current selection is also displayed in the status bar. The page number is calculated from the size the printer font selected in the Document Options command, plus syntax formatting options, and the settings made in Page Setup.

Cascade new windows

If checked, Source Insight will cascade new windows down the screen in the conventional Windows way. If not checked, Source Insight will position new windows exactly at the same location as the current window. The new window will cover the old window.

Maximize new windows

If checked, then Source Insight will automatically maximize newly opened windows. If not checked, then Source Insight will just open the windows in the normal MDI fashion.

Horizontal scroll bars for each new window

If checked, then each new source file window created will get a horizontal scroll bar.

Vertical scroll bars for each new window

If checked, then each new source file window created will get a vertical scroll bar.

Smooth scrolling windows

If checked, then windows will scroll more smoothly, instead of jumping one line at a time. Smooth scrolling only takes place if you are not scrolling continuously, but rather one or two lines at a time. If you start editing or scrolling quickly then smooth scrolling speeds up too. If not checked, then windows scroll one whole line at a time. You might want to turn this off if you do not have an accelerated video card and scrolling operations are slow with your video card.

Show line revision marks in left margin

If checked, then Source Insight displays a highlight in the left margin selection bar area next to each line that has been edited, or where lines have been deleted since the file was saved or opened. This makes it easy to see what lines you have edited. The Go To Next Change and Go To Previous Change commands (ALT+Keypad + and ALT+Keypad -) will jump forward and backward through the changes in the current file.

Show bookmarks in left margin

If checked, then Source Insight displays a bookmark icon in the left margin selection bar area next to each line that has a mark added with the Bookmarks command.

Show exact case of file names

If checked, then Source Insight displays file names using the exact upper and lower case of the name as it appears in the file system. If not checked, then Source Insight will format the file name to look a little nicer by converting it to lower case and capitalizing the first letter. Source Insight does not alter the file name if it already contains a mixture of upper and lower case letters.

This option only affects how Source Insight displays file names. The file names are stored internally and in the database exactly as the file system reports them.

Show read-only file names with ! mark

If checked, then Source Insight displays read-only file names in window titles with an exclamation point (!) at the beginning of the file name. This only affects how the file name is displayed in window titles.

Tile source and destination windows for Source Link commands

If checked, then Source Insight tiles two windows to show the source and destination source links whenever you use one of the source link commands, such as **Go To First Link**, or **Go To Next Link**. The source link commands are also used when you run a custom command that parses output for source links (such as a "compile" command). If not checked, then Source Insight will not alter the window arrangement when you used the source link commands, but it will active the window containing the target of the source link.

Source Insight does not perform tiling if the current window is maximized.

Use background textures

Enables the use of a gray texture in the background of user interface elements, such as toolbars and dialog boxes.

Use flat style toolbars

Uses the newer, "flat" style toolbar buttons, such as those in newer Windows programs. If not checked, then toolbar buttons will have a raised-button look.

Sort Window menu by usage and Sort Window menu by title

Use these radio buttons to specify the sorting order of the window names on the Window menu. If **Sort Window menu by usage** is selected, then the window names are sorted with the most recently used window at the top. If **Sort Window menu by title** is selected, then the window names are sorted alphabetically.

Styles...

Edits style properties.

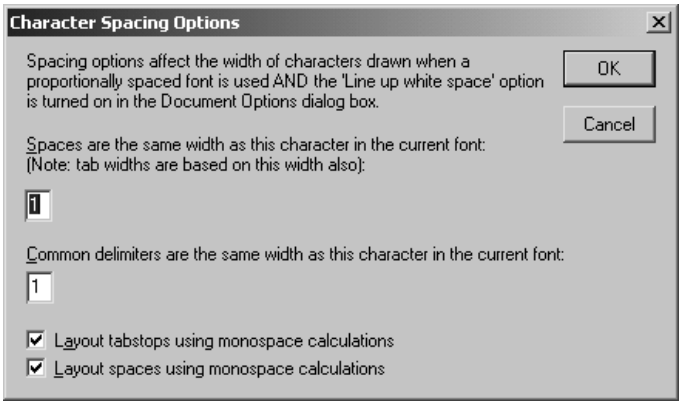
Spacing

Click this button to change character spacing options. See Character Spacing Options below.

Character Spacing Options

Character spacing options are useful with proportional fonts.

Character spacing options affect the width of characters drawn when a proportionally spaced font is used *and* the **Line up white space** option is enabled in the Document Options dialog box. If enabled, Source Insight will attempt to use a fixed width for spaces and tabs so that tabs line up the same way they do with a fixed pitch font. Programs generally look better with this turned on if you are using a proportional font.


Character Spacing Options
Spacing options affect the width of characters drawn when a proportionally spaced font is used AND the 'Line up white space' option is turned on in the Document Options dialog box.
OK
Cancel
Spaces are the same width as this character in the current font:
(Note: tab widths are based on this width also):
1
Common delimiters are the same width as this character in the current font:
1
Layout tabstops using monospace calculations
Layout spaces using monospace calculations

This dialog box lets you adjust how Source Insight computes the width of spaces, tabs, and common delimiters. These settings have no effect unless **Line up white space** is enabled in the Document Options dialog box.

The Space-Width Character

The space width character controls how wide a single space is, and therefore the displayed width of tab characters (tabs are some number of spaces wide). Source Insight computes a space to be same width as this character in whatever font is used for displaying. For example, the character "1" specified in this dialog box means that a space character will have the same width as the character "1". (Do not confuse the character with its numeric value 1. Source Insight does not interpret the character's numeric value.)

Source Insight computes the width based on the space width character so that spaces will scale correctly, independent of the font and the font size used.

Layout tabstops using monospaced calculations

This option controls how tab widths are displayed. If checked, then the width of a given tab will calculated assuming that you were using a monospaced font. This will generally make tabbed columns of text line up, even if you are using a variable pitched font for displaying your source code.

Layout spaces using monospaced calculations

This aligns space characters to appear how they would if a monospaced font is used. For example, 4 spaces in a row would appear the same width as a tab stop (if the tab width was 4 spaces). Source Insight looks at each line and tries to determine simply when to apply this rule to a space character. If it looks like you meant to line up columns manually using spaces, then it applies this rule. It only applies the rule for 2 or more consecutive spaces. Otherwise, it

calculates a space width to be the natural width of a space in the given font. This option is on by default.

Using this option, space size is natural, unless it looks like you meant to line up columns by using tabs and spaces. This is not an exact science!

Source Insight should be doing a good job of showing you how text lines up in a simple display, even if you are using Syntax Formatting. You can also use the Draft View command to see the simple text alignment.

Working with Wide Fonts

You may want to change this setting if you are working with a font that has unusual character widths, or if you just want to expand or contract your white space and indentation amounts. A narrower character will shrink the white space, and a wider character will expand it. This width is independent of the tab width setting specified in the Document Options dialog box because the tab width is specified as a number of fixed-width character columns.

The common delimiter character controls the width of delimiters in a way similar to spaces. The delimiters affected are - | \ / and ! which are typically narrow characters in most fonts.

Why all the Fuss?

You may be wondering, "Why go to all this trouble? Can't you make a tab stop be ½ inch or whatever?" The answer is a little complicated. The problem is that, unlike a word processor, Source Insight is trying to maintain a text file that other people may want to look at in a simple text editor.

Let's assume there are other people in your work group that don't use Source Insight, or that always use a fixed-pitch font for their source code. You don't want the pretty code you've edited in Source Insight to mess up the simple fixed-width tab stops when they look at the code.

A word processing program attempts to show text the way it would be printed on a physical printer. Source Insight is trying to show you how the text would look if you were looking at it in another editor in a fixed pitch font.

In a word processing program, text dimensions are measured in physical units, like inches or centimeters. It makes sense to have a tab stop at say, ½ inch. When the text is printed, the word processor makes sure the tab stop looks ½ inch wide on the printer too.

In Source Insight, tab stops are measured in fixed-size character columns. Source Insight tries to line up tabbed columns the same way it does with a fixed-pitch font.

If Source Insight just did the simple thing of moving to the next tab position, based on the horizontal pixel position, then when you look at the code with a simple fixed-pitch font, there may a different number of tabs than it appears on the screen.

Here is an example. Let's say somebody wrote this in Notepad, using Courier New (a fixed-pitch font), with tabs between columns so that X and Y, and Q and R line up. Both the words "narrow" and "wide" fit within column 0 - one tab stop, as shown below.

Tab stop:	0	1	2	3
Line 1:	narrow	X	Q	
Line 2:	wide	Y	R	

Now, in Source Insight, with rich formatting, "narrow" fits within a tab width, but "wide" doesn't. If Source Insight just pushed Y over by one more tab stop, this is what you get:

Tab stop:	0	1	2	3
Line 1:	narrow	X	Q	
Line 2:	**w i d e**		Y	R

Now Y is aligned with Q. The rest of the columns don't line up anymore. In fact, this is exactly what happens if you turn off **Line up white space** in Document Options.

When **Line up white space** is enabled, Source Insight tries to help the situation by lining up tab positions like this:

Tab stop:	0	1	2	3
Line 1:	narrow	X	Q	
Line 2:	**w i d e** Y		R	

If your code looks like it does above, then you may want to specify a different space width character, such as "M" or "W", which are wider letters in most fonts. This would have an effect like this, where all tab stops would be wider:

Tab stop:	0	1	2
Line 1:	narrow	X	Q
Line 2:	**w i d e**	Y	R

This also works the other way when the text looks a lot narrower than it would be in a fixed pitch font.

Document Options

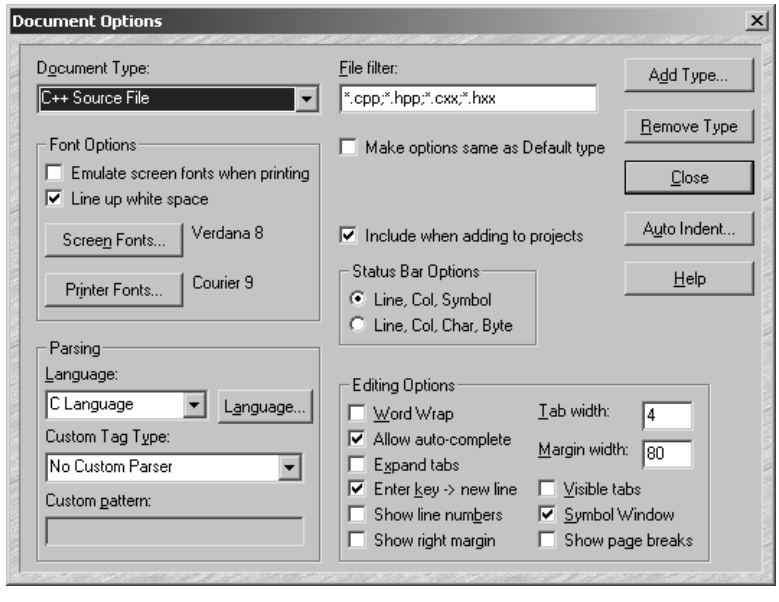
The Document Options command allows you to define editing and display options based on the file name or extension of the file you are currently editing.

Document Types

The document type determines the language and editing options for a file type.

A document type is a file classification that is defined in the Document Options command. Source Insight uses each file's name to determine what document type it has. Document types allow you to associates different types of source files with different behaviors in Source Insight.

Document Options Dialog box



Document Type

This pull-down list contains a list of all the document types you have defined so far. When you select a document type from this list,

the other text boxes in the dialog box are updated to reflect the properties of that document type. The first entry in the list is always the Default document type, which you may modify, but not remove.

When the Document Options dialog box first comes up, the document type of the current file is automatically selected here.

Add Type

Click this button to add a new document type. You will be prompted for the document type name.

Remove Type

Deletes the selected document type. This action is not undo-able.

Auto Indent...

Click this to change the Auto Indent settings for this document type. See below.

File Filter

This text box should contain a delimited list of file name specifications. The entries in the list can be delimited with a space, a semi-colon, or a comma. Each entry can be either an unambiguous file name, or an ambiguous wildcard file specification. The entries should not contain a drive letter or a full path that includes backslashes.

For example,

```
*.c;*.h
```

Files are matched to Document Types using wildcard filters.

Given a file name, Source Insight will identify the file's document type by searching all defined document types looking for a match on a file specification in the **File Filter** text box. In effect, Source Insight makes two passes through the entire set document type records to determine a file's document type.

1. First, it tries to find an exact match on the file name in all the File Filter lists.
2. If no exact match is found, it tries to find a wildcard match in all the File Filter lists.
3. If still no match is found, Source Insight assumes the file's type is the Default document type.

This means that you can treat some individual files in a special way. For example, "`*.inc`" is normally considered an Assembly File type, but let's say you have a file called "`cmd.inc`" that you want to have the C Source File type. In the C Source document type's **File Filter** text box, you would include "`cmd.inc`":

```
*.c;*.h;cmd.inc
```

If you include a simple * as the file filter for a document type, then it will become the default catchall type, instead of the "Default" document type. The catchall will only apply if the file does not already match any other document type.

Adding New File Extensions

You can add new entries to the file filter of a standard document type to make Source Insight include those files also. For example, by default, Source Insight considers C Source Files to be "`*.c`" and "`*.h`". If you also have C source files with .h2 extensions, then you can add "`*.h2`" to the **File Filter** list of the C Source File document type.

```
*.c;*.h;*.h2
```

You can control which document types are added to projects.

Whenever Source Insight displays a list of files, such as in the Add Files dialog box, the list is an expansion of the union of all File Filter text boxes in all defined document types. In other words, when you add new document types, those files will also appear in the file lists.

Make options the same as Default type.

If checked then the Editing Options and the Status Bar Options will be taken from the Default document type. This allows you to define many document types, but control their options from one location: the Default document type's record. The parser settings are not affected, and they remain unique to each document type.

If not checked then Editing Options and the Status Bar Options will be taken from the each individual document type record.

Include when adding to project

If checked, then the Add File command and the automatic add file feature will include this type of file when looking for new files to add to the project.

Font Options Group

Emulate screen fonts when printing

If checked, then Source Insight will attempt to select the same fonts for the printer as you have selected for the screen. If you are using a TrueType screen font, that should work fine. If not checked, then the font setting of the **Printer Fonts** button is used when printing.

Line up white space

This option only applies if you have selected a proportionally spaced font. Fixed-pitch fonts, such as Courier New, are not affected.

You can use a proportional font and still display indentation correctly.

If enabled, Source Insight will attempt to use a fixed width for spaces and tabs so that tabs line up the same way they do with a fixed-pitch font. Programs generally look better with this turned on if you are using a proportional font. If you are tired of using Courier New (or some other boring fixed-pitch font) to view your code, try this!

If disabled, then Source Insight uses the natural widths of characters as reported by the font.

This option is helpful if you have to work with other tools, or people that use tools that display source code with fixed-pitch fonts. You will be able to use a proportional font, and still keep a valid representation of the fixed pitch font indentation.

You can control how Source Insight computes the fixed width of spaces by clicking the **Spacing** button in the Preferences: Display dialog box.

For more information, see "Character Spacing Options" on page 215.

Screen Fonts

Click this button to select a font to use for displaying the file on the screen. The name of the currently selected font is displayed to the right of the button.

Printer Fonts

Click this button to select a font to use for printing the file. The name of the currently selected font is displayed to the right of the button. This setting only has an effect if the **Emulate screen fonts when printing** option (described above) is turned off.

Parsing Group

Language list

This pull-down list contains a list of all the languages defined in Source Insight. Select from this list to specify how Source Insight should parse and display the current document type. For example, to parse the document as a Java source file, select "Java Language" from the list. You can also select "None" to use no parser.

To add a new custom language, select the item <New Language>.

Language...

Click this button to open the Language Options dialog box. From there, you can add a new language, and edit the properties of a language. For example, you can edit the syntax formatting keyword list that is associated with each language. Each language type has its own keyword list. See also: "Language Options" on page 254.

Custom Tag Type

This pull-down list specifies what type of symbol is found as a result of using the custom parser pattern in the **Custom Pattern** text box. The list contains all of the possible symbol types. One of the entries in the list says "No Custom Parser". If that item is selected, then Source Insight does not use the custom pattern. If any other item is selected in the list, then the custom pattern should contain a regular expression pattern for parsing symbol names out of the file.

Custom Pattern

This text box should contain a valid regular expression with one group in it. The group describes what part of the matching pattern is assumed the symbol tag. The symbol parsed by using this pattern is given the type indicated by the **Custom Tag Type**. If the **Custom Tag Type** is set to "No Custom Parser", then this text box is ignored.

Using a custom pattern that allows you to parse some symbols out of files for which Source Insight has no built-in knowledge. For example, the following string parses sections out of .INI files like WIN.INI.

```
^\[\(.*\)\]
```

You can define a new document type named "INI File" that uses this custom parsing pattern. Now, when you open a file like WIN.INI, you can jump to any of the section names or see them all in the symbol window.

Status Bar Options

This group controls the appearance of the status bar at the bottom of the program window.

Line, Col, Symbol

The status bar shows the line number, the column number, and the name of the symbol that the insertion point is in.

Line, Col, Char, Byte

The status bar shows the line number, the column position on the line, the character position on the line, and the byte position in the file.

Editing Options Group

This group of items controls how the document type is edited. All files of the selected document type will have the following editing options in effect.

Word Wrap

If checked then Source Insight will automatically wrap words onto the next line when the insertion point moves past the margin width. This only applies while you are typing new text. If not checked then Source Insight will not do any automatic wrapping of text.

Allow auto-complete

If checked, then symbolic auto-completion is allowed for the document type if auto-completion is enabled globally in the Options > Preferences: Typing dialog box.

Tab Width

The width of a tab character in character spaces.

Margin Width

The width of text in characters spaces before automatic word wrapping will occur

Expand tabs

If checked then Source Insight will expand a tab character to the equivalent number of spaces when you type a tab. The text will look the same as if a tab was typed, but spaces will be used to fill. If not checked then Source Insight will simply insert a tab character into the file when you type a tab.

Enter key inserts new line

If checked then pressing the Return or Enter key while typing will insert a new line. If not checked then pressing Return or Enter will move the insertion point to the beginning of the next line.

Show line numbers

Displays line numbers in the left margin.

Show right margin

Displays a light vertical line at the right margin. If you are using proportional fonts, then the right margin position is only approximate.

Visible Tabs

If checked then Source Insight will display a special symbol where ever a tab characters is, instead of just displaying white space.

Symbol Window

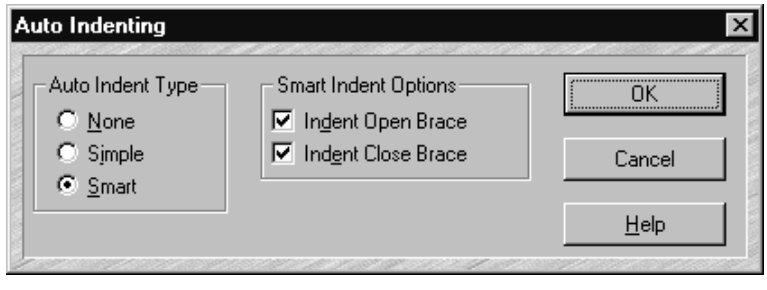
If checked, then files with this document type will have a Symbol Window attached at the left side of their source windows.

Show page breaks

Source Insight will show light horizontal lines that represent the printed page breaks. The pagination is computed based on syntax formatting, and the printer font that is selected.

Auto Indenting

The auto-indenting feature controls the level of indentation as you type new text. Source Insight supports **Simple** and **Smart** types of auto-indentation. Not all languages support the Smart level.



Auto Indent Type

Specifies the type of auto-indenting. Automatic indenting occurs when you insert new lines.

- **None.** No special indenting occurs. Source Insight will return the insertion point to the very beginning of the next line when you insert a new line or word wrap.
- **Simple**. Source Insight will automatically indent text to line up with the previous or following line.
- **Smart**. Source Insight will automatically increase or decrease the indentation level when you insert new lines.

Not all languages support smart indenting. If this button is selected, then the Smart Indent Options are applied.

Smart Indent Options

These check boxes determine how the smart indenting affects open and closing curly braces.

Desired Format	Check box setting

if (x)
{
}
	Clear both boxes.

if (x)
{
}
	Select both boxes

if (x) { }	Select **Indent Open Brace**; Un-select **Indent Close Brace**

Draft View

The Draft View command on the View menu toggles the draft view mode on and off. When draft mode is on, almost all syntax formatting is suppressed, except for color changes. All text is displayed using the "Draft View" style, which can be edited with the Style Properties dialog box.

The "Draft View" style is preset to use a monospaced font (Courier New).

The Syntax Formatting features of Source Insight are powerful, but sometimes you need to see how text will line up in another editor or in a simple display mode when only a single font is used. Draft mode is useful for quickly switching your display to a basic monospaced font display. This is particularly useful if you are using spaces instead of tab characters to line columns up.

When draft mode is active, it overrides the settings of the Preferences: Syntax Formatting and Syntax Decorations dialog boxes.

Drag Line Down

Moves selected text down by one line. This is useful for dragging a whole line or lines down below something else in a file.

Drag Line Down More

Moves selected text down by several lines. This works like the Drag Line Down command, only it moves the lines further.

Drag Line Up

Moves selected text up by one line. This is useful for dragging a whole line or lines up above something else in a file.

Drag Line Up More

Moves selected text up by several lines. This works like the Drag Line Up command, only it moves the lines further.

Duplicate

The Duplicate command creates a duplicate of whatever is selected.

Duplicate Symbol

(On the Symbol Window right-click menu)

The Duplicate Symbol command creates a duplicate of the selected symbol.

Edit Condition

Use this command to edit the value of a selected parsing condition. This is used for languages that support conditional compilation, such as C, C++, and Window Resource files. Conditional code is placed between #-directives such as `#ifdef`.

Source Insight can parse those sections of code conditionally depending on the value of conditions that you specify. The Edit Condition command lets you edit the value of a condition, or edit the list of conditions.

To use this command, select a word that is a condition in your code. Then select **Edit Condition**. You will be able to specify that condition's value.

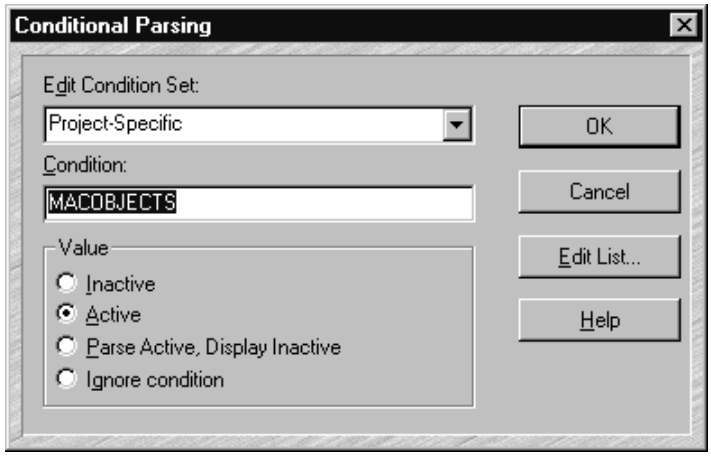
For example, place the cursor inside of `MACOBJECTS` and select **Edit Condition**.

```
#ifdef MACOBJECTS
int jklm;
#endif
```

Project vs. Global Conditions

There are two condition sets. One is project specific and is stored with your project. The second set is global and applies to all projects. If a condition appears in both lists, the project specific value is used.

Edit Condition Dialog box



Edit Condition Set

Selects which condition set you want to affect. This is either the project-specific, or the global list. Any changes you make to the condition's value are put into the list you select here.

Condition

The name of the condition identifier.

Value

The value of the condition. Each condition can have one of the following values:

- **Active** - the condition is assumed true.
- **Inactive** - the condition is assumed false.
- **Parse Active, Display Inactive** - the condition is assumed to be true, but code inside the conditional block is formatted with the Inactive Code style.
- **Ignored** - the condition is unknown. Source Insight ignores the conditional directive altogether. This is the default case for any conditional directives encountered. An ignored condition is, by definition, not in the condition set.

Edit List

Click this to go to the **Conditional Parsing** dialog box, which displays all the defined conditions. The list you get to edit depends on which one is selected in the **Edit Condition Set** control.

End of Line

The End of Line command moves the insertion point to the end of the current line.

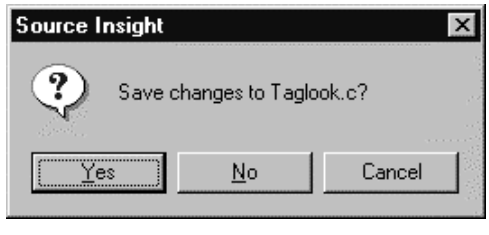
End of Selection

The End of Selection command moves the insertion point to the end of an extended selection. If the current selection is not extended, this command does nothing.

Exit

The Exit command exits the Source Insight program. Source Insight will ask you if you want to save each file that has been changed but not saved.

If you have unsaved files open when you quit, a dialog box will appear with the following buttons:



Yes

Click this button to save the file and proceed to quit.

No

Click this button to not save the file and proceed to quit.

Cancel

Click this button to abort the exit. The file will not be saved. Source Insight will resume as though you did not ask to exit. All files will still be open, and some may not have been saved.

When Source Insight exits, it saves the current workspace file, so you may resume your session when you run Source Insight the next time.

Exit and Suspend

The Exit and Suspend command writes a Source Insight recovery file out and then closes Source Insight without saving any files. This allows you to exit Source Insight and save your edits without altering any files. Run Source Insight again to recover your edits.

Warning: do not alter the files that Source Insight had open before you run Source Insight a second time to recover your changes. Source Insight's recovery system relies on the original files being left unchanged.

Expand Special

Used inside tree lists, this expands the selected item a specified number of tree levels.

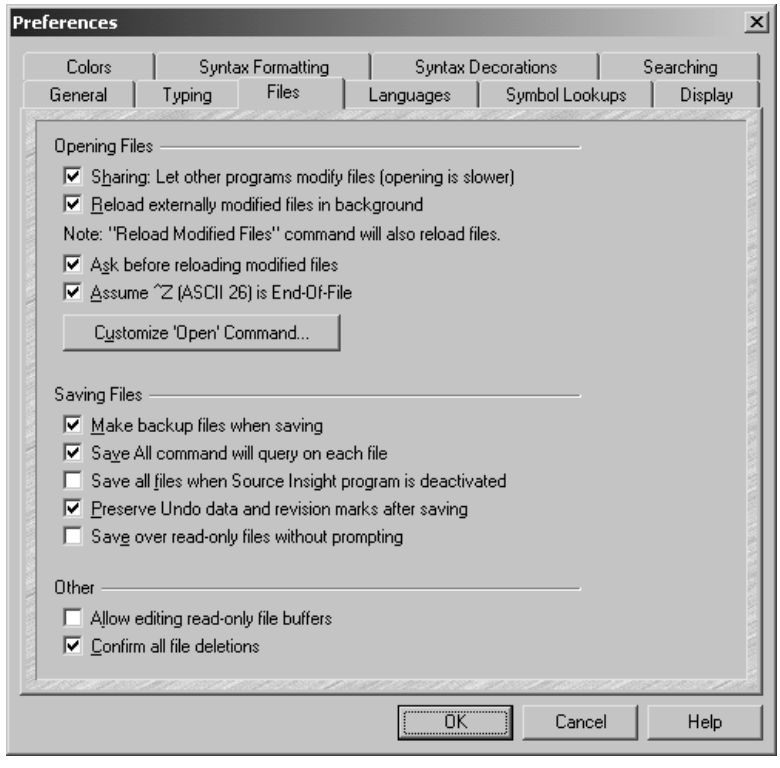
Number of levels to expand

Type the number of levels, beyond the selected node, that you want to expand the tree. For example, if you type "1", then one level below the selected node is revealed.

File Options

This command activates the File page of the Preferences dialog box. It allows you to set file loading and saving options.

File Options Dialog box



Sharing

Enable this to allow other programs to modify the files that are open in Source Insight. In other words, a file that is open in Source Insight can be written over by another program. Turning this on will cause Open and Save operations to be a little slower, and it will use more disk space for each open file.

Disable this to give Source Insight exclusive write access to the files. Files will still be opened for Read-Only, except during Save operations. However, *Deny-Write* sharing will be used. This is also faster.

Reload externally modified files in background

Turn this on to enable Source Insight to detect that files have been modified externally. Files are reloaded automatically. Files are

checked every few seconds and whenever the Source Insight application window comes to the front. Each modified file is reloaded silently, without any prompting, unless the **Ask before reloading modified files** option is on, or you have edited the file in Source Insight.

If disabled, you can still run the Reload Modified Files command to manually force Insight to reload any modified files that are found.

Ask before reloading modified files

If enabled, then Source Insight will prompt you when it detects that a file has been modified externally to see if you want to reload it.

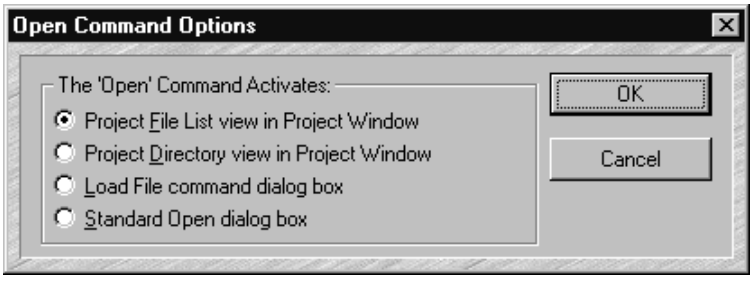
If disabled, then Source Insight will automatically reload externally modified files silently without any prompting, unless you have also edited the file in Source Insight. If you have already edited a file, the reload operation is not performed.

Assume ^Z (ASCII 26) is End-Of-File

If enabled, then Source Insight will stop loading a file when it sees the EOF (ASCII 26) character. When it saves the file, an EOF character will be appended to the end. If disabled, then Source Insight will continue to read past the EOF character.

Customize the 'Open' Command...

This lets you pick the action performed by the Open command. The Open command is normally assigned to Ctrl+O, and it has a toolbar button.



Make backup files when saving

If checked, then Source Insight will move the previous version of a file on disk to the backup directory whenever it saves the file. The backup directory is stored in a subdirectory named "Backup" in the Source Insight program directory.

If not checked, then Source Insight will save files without preserving the previous version on disk.

Save All command will query on each file

If checked, Source Insight will ask you if you want to save each modified file when running the Save As command. You will have an opportunity to save it, not save it, or cancel the Save As operation on each file.

If not checked, Source Insight will go ahead and automatically save all files that have been modified since the last time they were saved.

Save all files when Source Insight program is deactivated

If checked, Source Insight will automatically perform a "Save All" command when the Source Insight application window loses focus (i.e. every time you activate a different application). This enables you to work with another editor or IDE that has the same files open. For example, if you switch to your IDE application then Source Insight will automatically save all edited files to disk. If not checked, Source Insight will not save files when deactivated.

Save over read-only files without prompting

If checked, then Source Insight will save over a read-only file without warning you it is read-only. Actually, you will be warned the first time you attempt to save over a read-only file in a session.

If not checked, then Source Insight will prompt you for each read-only file that is saved. You will still have the option of overwriting a read-only file on a file-by-file basis.

When Source Insight saves over a read-only file, the file is changed to read/write.

It is not recommended that you enable this option, as you may accidentally write over valuable files that are read-only for a good reason. This option may be of use if your source control system will respect a read/write file as “checked out”. Thus, you can edit and save files before you check them out.

Preserve Undo and revision marks after saving

If enabled, then you will be able to perform Undo, and see revision marks, even after you save a file.

Allow editing Read-Only file buffers

This option lets you to edit a file buffer, even if it is marked read-only. You will not be able to save back to the file, unless you change its permissions to read/write outside of Source Insight, or you explicitly overwrite the read-only file by clicking the “Overwrite” button during saves. Depending on your source control system,

you may be able to check out your version of the file, and then return to Source Insight and save the file.

Note that when you edit a file buffer inside Source Insight, you are not changing the file on disk until you use the Save command, or you cause Source Insight to save the file some other way, such as task switching out of Source Insight to another program when you have the **Save all files when Source Insight is deactivated** option enabled.

Confirm all file deletions

Source Insight will confirm before deleting any source files. Source Insight does not delete source files when you remove them from a project, or when you delete a project. It only deletes the project data files created by Source Insight. However, you can select a source file in the Project Window and delete it.

Function Down

The Function Down command moves the insertion point to the next function or method defined in the current file.

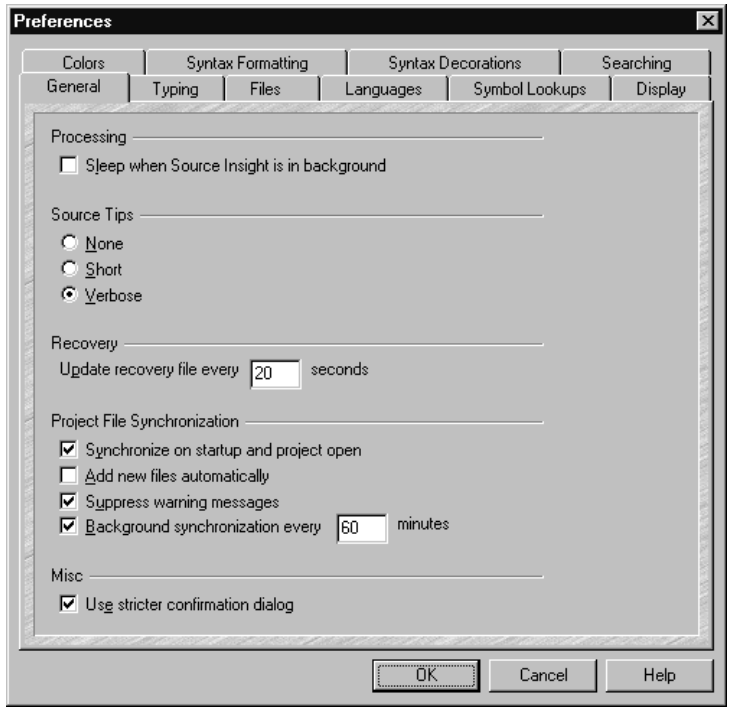
Function Up

The Function Up command moves the insertion point to the previous function or method defined in the current file.

General Options

This command activates the General page of the Preferences dialog box. It allows you to change miscellaneous Source Insight options. These options are part of the current configuration.

General Options Dialog box



Sleep when Source Insight is in background

If checked, Source Insight will not perform any background processing when the Source Insight application is minimized or not in front (such as when a custom command is in front).

If not checked, then background processing will occur normally. However, Source Insight lowers its priority to below normal when the program in the background.

Background Tasks

Source Insight performs a number of tasks in the background, including the following:

- It parses files and updates the Symbol Window contents for all open source windows.
- It checks for finished Custom Commands.
- If **Background Synchronization** is enabled, then it synchronizes all the files in the project, possibly adding new ones in the process.
- It synchronizes the Context Window with the current text selection.
- It synchronizes the Relation Windows with the current text selection.
- It checks for and reloads open files that are modified outside of Source Insight by other programs.

Usually there is nothing to do, in which case Source Insight sleeps and uses little or no processor cycles.

Source Tips

Sets the level of information provided by pop-up source tip windows. Source tip windows appear when you hover the mouse cursor over a symbol identifier for a few seconds.

Crash Recovery

Recovery: Update recovery file every NNN seconds

Specifies how often Source Insight will update the crash recovery file. The default value is 15 seconds. The recovery file is only updated when there have been edits since the last time it was updated. The recovery update is very fast and you probably will not even notice anything being saved. You will never be interrupted to save the recovery file. You should keep this interval short.

Project File Synchronization

Synchronize on start-up

If checked and background synchronization is turned on, then Source Insight will check file time-stamps right away every time you start Source Insight or open a new project. When files are found to be out of date with respect to the project, Source Insight marks the files for re-synchronizing. Later, the files will be rescanned and synchronized in the background.

Add new files automatically

If enabled, then before synchronizing all the files, Source Insight will add new files in the project's root directory and in all subdirectories, recursively. However, only directories that already have project files in them are scanned. Directories that are not descendants of the project root directory are not scanned. This feature allows you to simply add new files to your project directories on disk, such as with a source control system, and then have Source Insight add those new files to your Source Insight project automatically.

Background project synchronization every NNN minutes

If enabled, then Source Insight will perform the actions of the Synchronize File command in the background, while you edit. You typically will not have to run the Synchronize Files command at all if this option is enabled.

NNN specifies how often the project files should be examined to determine if files need re-synchronizing. When this period expires, Source Insight checks the file time-stamps and begins the synchronization process in background.

Use stricter confirmation dialog

If checked, then when Source Insight confirms an operation, you will be required to type "yes" to confirm it.

Go Back

The Go Back command moves the insertion point to its previous location. Source Insight keeps a selection history, which is a circular list of the last 100 positions you've visited. The selection history is global to all open files, not just the current file.

If you have used an Internet browser with Back and Next buttons, you will be familiar with the Go Back and Go Forward commands in Source Insight.

Using Go Back to View a Function Call Chain

The Go Back command works nicely with the Jump To Definition command. If you jump to a function definition, you can use Go Back to go back to the function caller. This process can recurse many times. You can use Go Back, and Go Forward to traverse the call chain forward and backward.

The selection history is circular, so eventually you will end up at your starting point.

You can use the Selection History command to show the list. That command shows each position, along with the function or enclosing symbol at each location.

Go Back Toggle

The Go Back Toggle command toggles between running the Go Back command and the Go Forward command. Using Go Back Toggle repeatedly will toggle you between your last two positions.

Go Forward

The Go Forward command moves the insertion point to the next location in the selection history, which is a circular list of the last 100 positions you've visited. See the *Go Back* command for more details.

Go To First Link

The Go To First Link command locates the first source link and does the following:

- It selects the link line in the link source file.
- It selects the link line in the link target file.
- It ensures both files are visible on the screen in windows. If the current window was maximized when this command was used, then only the link target file will be made visible.

The Go To Next Link and Go To Previous Link commands do the same thing, except with the next and previous source link, respectively.

First Source Link

The "first source link" is the first link in the link source file with which a Go To Link Location command was used. For example, if you used the Search Files command to create a Search Results file containing source links, and then you used the Go To Link Location command on a line in the Search Results window, the first source link is determined to be the first link in the Search Results window.

The Go To First Link, Go To Next Link, and Go To Previous Link commands are used to quickly skip from link to link, and are especially useful when the source links are connecting compiler error messages and program source lines.

Using Links With Compiler Errors

If you spawn the compiler from Source Insight, using a custom command, and the output is captured and parsed for error messages, then you can use Go To First Link and Go To Next Link to view each error in your source files.

When you define the "Compile File" custom command, you should have the "Parse Source Links" option on. Source Insight will then search the compiler output and setup source links for each error message. In this case, the "link sources" are each error message in the compiler output file. The "link target" for each link is the file and line number given in each error message.

To view source lines with errors

To run a build or compile command, and let Source Insight position to each error message:

1. Run the "Compile File" or "Build Project" custom command, which is defined as described above.
2. Assuming there are errors, when the compiler finishes the error messages will be in a command output window. Source Insight will automatically setup the source links and run the Go To First Link command. The first error message and the erroneous source line will be selected and made visible.
3. Run the Go To Next Link command. The next error message in the command output window is selected, and the target of that link is shown, as was the first error.
4. Continue to use the Go To Next Link command until all the links (error messages) have been visited. If there are no more links, then Source Insight beeps and the message, "No links." will appear in the status bar.

Using Links With Search Output

The Search Files command puts its output into a Search Results window. Along with each line of text in the Search Results window is a source link. In this case, the "link sources" are each line in the Search Results window. The "link target" for each link is the file and line where the search pattern was found.

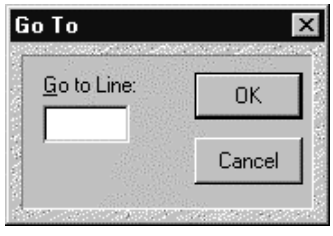
To view each place where a pattern was found:

To perform a search and then visit each place where the pattern was found:

1. Run the Search Files command.
2. Use the Go To First Link command to see the first match.
3. Use the Go To Next Link command to see successive matches.
4. Continue using the Go To Next Link command until the "No Links." message appears.

Go To Line

The Go To Line command allows you to type a line number and position the insertion point on that line.



Go to Line

Type the line number here.

OK

Click OK to go to the line number. If you type a line number beyond the end of the file, Source Insight positions to the last line in the file.

Cancel

Click Cancel to cancel the Go To Line command.

Go To Next Change

Moves the cursor to the next block of lines that were edited. It moves the cursor to the next set of change marks.

Go To Previous Change

Moves the cursor to the previous block of lines that were edited. It moves the cursor to the last set of change marks.

Go To Next Link

The Go To Next Link command behaves the same as the Go To First Link command, except the next link is used. See Also: Go To First Link command.

Go To Previous Link

The Go To Previous Link command behaves the same as the Go To Next Link command, except the previous link is used. See Also: Go To First Link command.

Help

The Help command brings up help on Source Insight. You can also press F1 while a dialog box is up and Source Insight will display help on the current command.

Help Mode

The Help Mode command turns on the help mode. When a command is invoked while the help mode is on, Source Insight displays help on the command and turns off the help mode, instead of running the command. You can cancel the help mode by running the Help Mode command again.

For example, to get help on the File > Open command, type Ctrl+F1 to turn on the help mode. A message will appear in the status bar to indicate that help mode is active. Now go to the File menu and select the Open command. A help window will open and display help on the Open command.

You can also press F1 while a dialog box is up and Source Insight will display help on the current command.

Highlight Word

Toggles word-highlighting for the word under the cursor in all source windows. This is like using a highlighter pen on paper. If you select a word, and use the Highlight Word command, then anywhere that word appears in your source, it will appear highlighted.

By default, the highlight appears like bold black text with a bright yellow background. However, you can set the highlight effect

yourself by editing the Highlight style. The Style Properties command is used to edit styles.

Incremental Search

The Increment Search and Incremental Search Backward commands invoke the incremental search mode. By default, F12 is the Incremental Search command, and Shift+F12 is Incremental Search Backwards.

Once in incremental search mode, Source Insight will start finding matches as you type characters, starting at the current cursor position. As you type more characters, the search will become more specific. The characters you type will appear at the bottom in the status bar.

You can exit the incremental search mode with any command key (such as an arrow key) or by pressing Esc. If you want to search again, just type F12 twice - it will load the old pattern and find again.

Incremental Search Mode

Once you press F12, you enter the incremental search mode. Once in the incremental search mode:

- F12 will search again for the current string, or it will load the current string with the previous one if the current string is empty.
- Shift+F12 will search backwards.
- Backspace reduces the search string.
- Esc will cancel and return to the initial position.
- Enter will stop and leave the selection at its current position.
- Any key that maps to a command will exit the incremental search mode and leave the selection at its current position, and then the command will execute.
- Any other simple key is added to the current search pattern, which is displayed at the bottom in the status bar.
- The search buffer is left with the last successful search pattern.

The incremental search pattern is not case sensitive, unless you type an uppercase character.

Incremental Search Backward

Searches backwards, incrementally, in the current file. See "Incremental Search" on page 242.

Horizontal Scroll Bar

This command toggles the horizontal scroll bar on and off in the current source file window

HTML Help

Looks up the currently selected word in the HTML Help file. The HTML Help file is the one specified in the Setup HTML Help command. See "Setup HTML Help" on page 340.

Indent Left

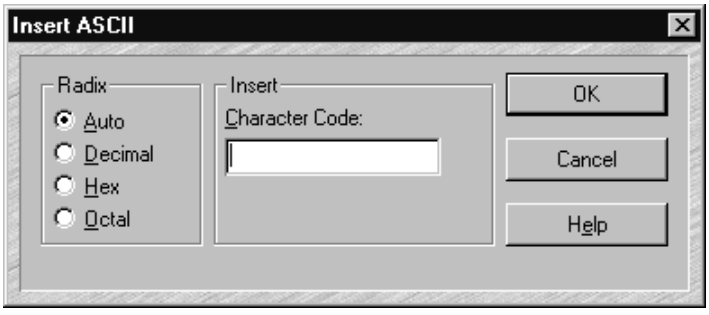
The Indent Left command outdents the lines intersecting with the current selection to the left by the size of one tab stop. Lines that begin with # are not indented.

Indent Right

The Indent Right command indents the lines intersecting with the current selection to the right by the size of one tab stop. Lines that begin with # are not indented. Source Insight indents lines to the right by inserting a tab character. If you have the "Expand tabs to spaces" option on for the current document type, then spaces equivalent to a tab stop are inserted instead of a tab character.

Insert ASCII

The Insert ASCII command inserts a character that you specify with its ASCII code.



Radix

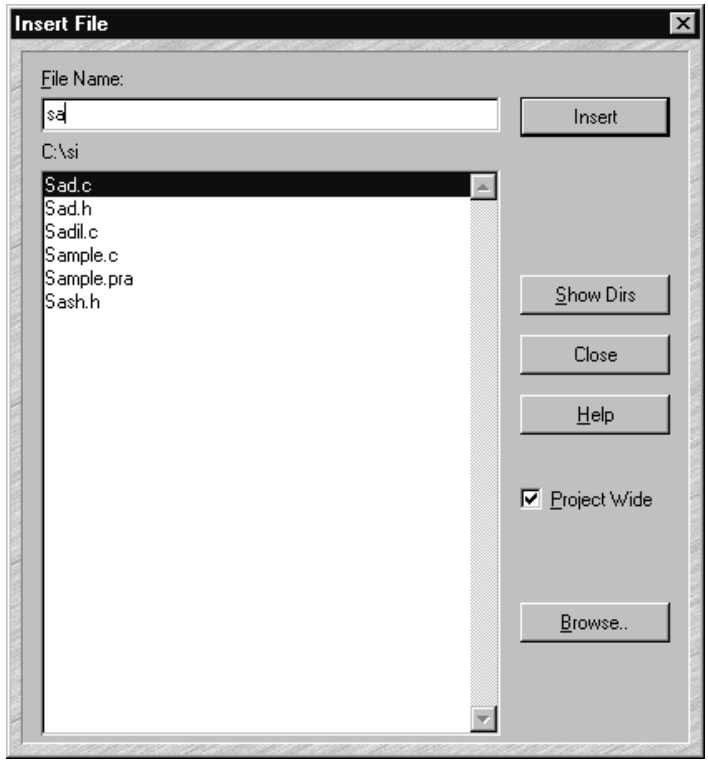
Selects the input radix of the character code you typed. If **Auto** is selected, then the input radix is determined from the text you type. For example, if you type 0x20, then the radix is assumed to be Hex, and ASCII 32 is inserted.

Character Code

This is the ASCII code of the character you want to insert.

Insert File

The Insert File command pastes the text of another file into the current selection.



File Name

The name of the file to insert. You may also type a series of wildcard specifications and click the **Insert** button and Source Insight will replace the file List contents with the results of wildcard expansion. The wildcards are expanded in the current directory.

File list

If the **Project Wide** option is enabled, then this list displays all files in the current project. If disabled, then this list displays all the files in the current working directory. The current directory path is displayed at the top of the list box.

Insert

Click Insert to insert the contents of the file in the File Name text box. If the File Name text box contains one or more wildcard specifications, then Source Insight will replace the File List contents with the results of the expansion. The wildcards are expanded in the current directory.

Show Dirs

Click this button to toggle the list box contents between showing file names, and showing only subdirectory names.

Browse

Click this button to bring up the standard Windows Open dialog box, which allows you to browse around your disks.

Project Wide

If checked, the file List will show all the files that have been added to the current project. If not checked, the file List will show files in the current directory only. This option is always unchecked if no project is open.

Insert Line

The Insert Line command inserts a new, empty line before the line the selection is on. The cursor does not have to be at the beginning or end of the line.

If Auto Indent is turned on, then the new line will be indented even with the line below it.

Insert Line Before Next

The Insert Line Before Next command inserts a new, empty line after the line the current selection is on. If Auto Indent is turned on, then the new line will be indented even with the line above it.

Insert New Line

The Insert New Line command inserts a new line starting at the insertion point. This is just like pressing Enter, except the cursor does not move to the next line.

Join Lines

The Join Lines command joins the line the insertion point is on, and the next line, so that it forms one single line. If the selection is extended, then all the lines intersecting with the selection are joined.

Jump To Base Type

Moves the cursor to the most base structure type of the selected variable or type. For example, consider the following code:

```
struct MyStruc
        {
        int afield;
        int anotherfield;
        };

// MS type is defined as struct MyStruct
typedef struct MyStruct MS;

MS ms;  // declare ms with a type of "MS"
x = ms.afield;
```

If you put the cursor in `ms` in the assignment statement (or anywhere the `ms` variable appears), the Jump To Base Type command will jump to the definition of `struct MyStruc`, because that is the most base structure type of the variable. It won't stop at the `typedef` of `MS`.

Jump To Caller

Jumps to the caller of the selected function, if any. For example, if you put the cursor on a function name and use Jump To Caller, then you will jump to the function that calls it. If more then one function calls it, you will see a list from which you may pick.

Jump To Definition

The Jump To Definition command takes the symbol from the first word in the current selection and jumps to its definition. The Go Back and Go Forward commands are useful for going back and forth between all your jumping spots.

To use this command:

1. Select within the symbol name as it appears in a source file.
2. Type Alt+= to jump to the actual symbol definition.

Mouse Shortcut

In the default configuration, you can also use the mouse to easily invoke this command. Pointing at a symbol in a file and performing Ctrl+Left Click performs the Select Word command. Ctrl+Left Double-Click then performs the Jump To Definition command.

To use this command with the mouse, point and double-click at the symbol name with the left mouse button while holding the Ctrl+key down.

Opening Header Files

You can also use the Jump To Definition command when the cursor is in a file name, such as in an #include statement to open the file.

Jump To Link

Moves to the other end of the source link at the current line. See Also: "Go To First Link" on page 238.

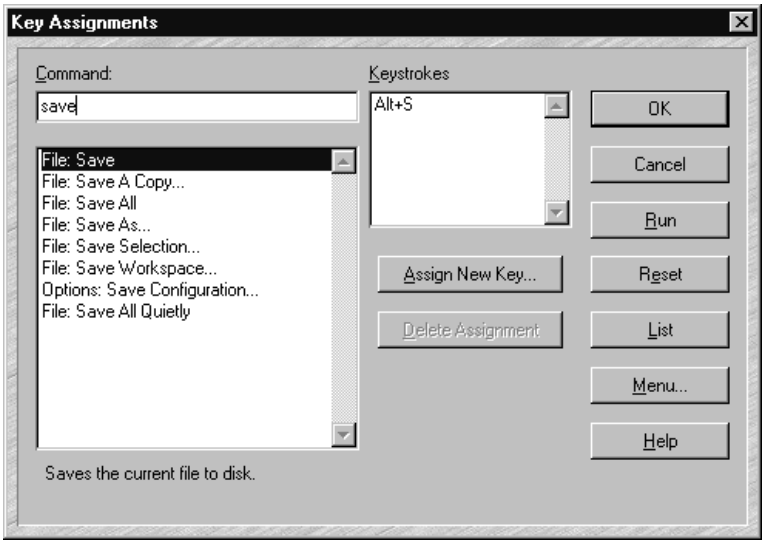
Jump To Prototype

Jumps to the declaration of the selected function's prototype. This only works if the selected symbol is a function.

Key Assignments

The Key Assignments command allows you to assign or reassign keystroke combinations to commands. The mouse buttons can also be assigned to commands. The key assignments are part of the current configuration.

Key Assignments Dialog box



Command

You can type into this text box to narrow down the command list, so that you can find the command you want easily. With syllable matching, you can simply type a word contained within any command name.

Command list

Lists all the Source Insight commands, including macros and custom commands that you've defined. When you select a command here, the keystrokes list is loaded with all the keystrokes currently assigned to the selected command.

Keystrokes list

Lists all the keystrokes assigned to the selected command. Select a keystroke here before clicking the **Delete** button when deleting it.

OK

Click this to record the new key assignments in the current configuration.

Cancel

Click this to cancel the command. The current configuration will not be affected by any changes in the dialog box so far.

Assign New Key...

Click to add a new keystroke or mouse click to the command selected in the **Command list**. See below.

Delete Assignment

Click to remove the assignment of the keystroke selected in the **Keystrokes list** from the command selected in the Command List.

Run

Click to run the selected command. This also records any changes you have made.

Reset

Click to reset the key assignments to their default, factory settings. Source Insight will ask you if you are sure you want to do this.

List

Click to create a key assignments list file. This also records any changes you have made. The list file is just a text file that contains a list of commands, and their key assignments.

Menu

Click Menu to record the new key assignments in the current configuration, and then run the Menu Assignments command.

Assigning Keystrokes

You can add any combination of Alt, Ctrl, and Shift key modifiers with any other key, including mouse buttons.

To assign a new keystroke combination to a command:

1. Select the command in the **Command list**.
2. Click the **Assign New Key** button.
3. Type the keystroke(s) that you want to assign. Pressing Esc cancels the assign procedure. If the keystroke you typed is already assigned to a different command, Source Insight will ask you if you want to re-assign it.

Assigning Mouse Clicks

To assign a mouse click to a command:

1. Select the command in the **Command list**.
2. Click the **Assign New Key** button.
3. Click the mouse button that you want to assign. If you want a modifier key, such as Alt, Shift, or Ctrl, to be included, press the modifier key before clicking the mouse button. You can even use the Left mouse button to modify the right button. Pressing Esc cancels the assign procedure.

Deleting a Key Assignment

To delete a keystroke assignment from a command:

1. Select the command in the **Command list**.
2. Select the keystroke to be deleted in the **Keystroke list**.
3. Click the **Delete** button.

Keyword List

Brings up the Language Keywords dialog box, which lets you edit the language keywords used for syntax formatting in the current language.

The Language Keywords dialog box lists keywords and styles. The title of the dialog box will specify which language type you are working with. Source Insight uses a very fast hashing technique to maintain large keyword lists and still have outstanding performance.

Keywords and Styles

The keyword list contains all the language keywords that can be highlighted with syntax formatting. Each keyword in the list is

associated with a syntax style. The Style Properties command is used to set the formatting options of each style.

For example, in the C Language keyword list, the word "NULL" is associated with the "Null Value" style.

To determine the formatting of any given word in a window, Source Insight locates the word in the keyword list of the appropriate language type. The keyword list contains a style name, which in turn implies the formatting associated with the style.

Therefore, starting with a file name and a word in the file, Source Insight derives the word's style with these relationships:

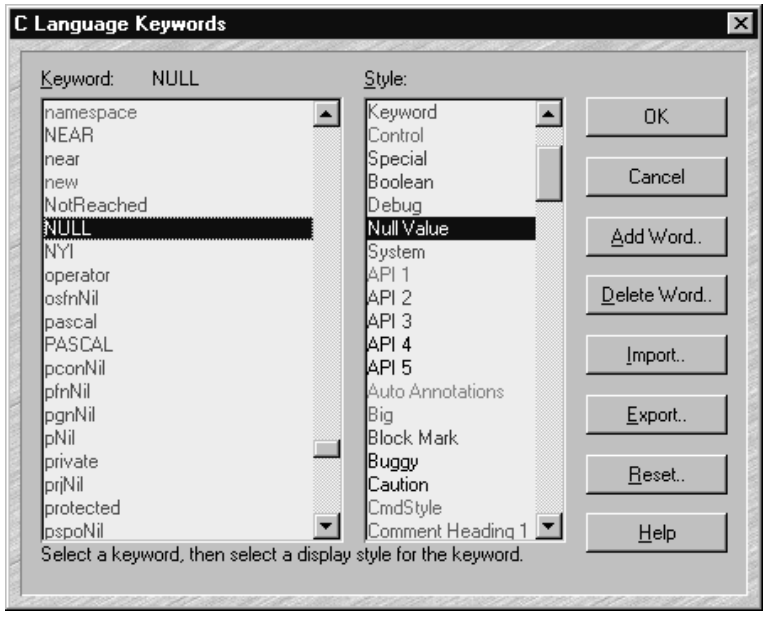
```
File name >> document type >> language type
```

And:

```
word + language type >> keyword entry >> style name >>
formatting
```

By having keywords assigned to formatting styles, you are able to change the syntax formatting quickly by simply changing the style with the Style Properties command. Then all language keywords associated with that style reflect the new style formatting.

Language Keywords Dialog box



Keyword

The keyword list for the language. When you select a keyword from the list, the keyword's associated style is selected in the **Style** list.

Style

The list of all syntax formatting styles. When you select a style from this list, you are changing the style associated with the keyword selected in the keyword list. You can also double-click on a style name to edit the style.

OK

Click OK to record your changes.

Cancel

Aborts the command and ignores your changes.

Add Word

Click this button to add a new word to the keyword list. You can type any single word that does not include spaces. Source Insight will add the word to the keyword list. After adding the word, make sure you select the style you want it to have.

Delete Word

Click this button to delete the word currently selected in the keyword list.

Import

Click this button to import new keyword list entries from an external text file. See below for more information.

Export

Click this button to export the keyword list to a text file.

Reset

Click this button to return the language keyword list to the factory default settings.

Importing and Exporting Keyword Lists

The **Import** and **Export** buttons in the Language Keyword dialog box allow you to import and export keyword-style associations from and to text files.

The text file should contain keyword, style name pairs; one per line:

```
<keyword> , <style-name>        or
"<keyword>", "<style-name>"
```

Each keyword should be a single word without white space. The style name should be one of the defined style names that are listed in the Language Keywords style list, or the Style Properties style list. You may enclose the keyword or style name in double quotes.

When importing, duplicate keywords are ignored.

Import Options

When you click the **Import** button in the Language Keywords dialog box, you will be given the option of either replacing or merging the keyword list.

Replace current keyword list with imported list

Select this if you want to completely replace the keyword list currently loaded with the imported list. Note: even after importing a list, you can still click the Cancel button in the Language Keywords dialog box to ignore the changes you made to the keyword list.

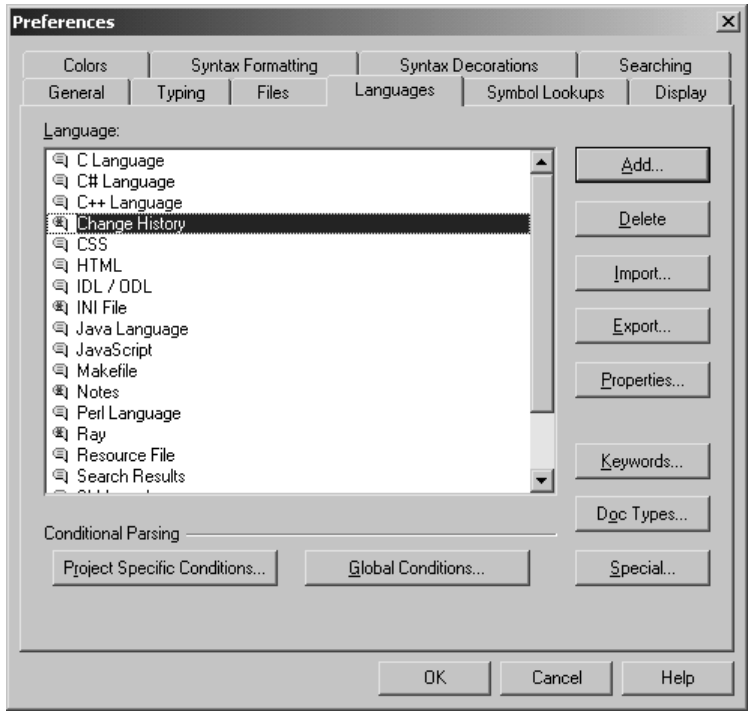
Merge current keyword list with imported list

Select this if you want to merge the imported list with the currently loaded list. Merging means that the imported list will be added to the existing list, and imported keywords will replace like-named keywords in the current list.

Language Options

This command activates the Languages page of the Preferences dialog box. It allows you to edit language-specific options, such as the language keyword list.

Source Insight supports two types of languages: Built-in and Custom. You can alter a few options for built-in languages. For custom languages, you can control all the parameters for a generic language.



Language

This list contains all the installed languages. Custom Languages are marked with a red asterisk in the icon.

You associate a language with a particular document type with the Document Options command. See "Document Options" on page 219.

Add...

Click this button to add a new custom language.

Delete...

Click this to delete the selected custom language. Only custom languages can be deleted. The built-in languages cannot be deleted.

Import...

Click this to import a custom language into the list from a custom language file. A custom language file (.CLF) contains all the properties of a single custom language.

Export...

Click this to export the selected custom language to a custom language file (.CLF). A custom language file contains all the properties of a single custom language. You can export a custom language so that other people can import the language into their Source Insight configurations.

Properties...

Click this button to open the Language Properties dialog box.

Keywords

Click this button to edit the keyword list associated with the selected language type. The Language Keywords dialog box will appear.

Doc Types...

Opens the Document Options dialog box.

Special...

Displays options that are specific to the selected language. Not all languages have special options.

Conditional Parsing

Project Specific Conditions...

Click to edit the conditions defined that are specific to the current project only. These conditions are only in effect when the current project is open, and only for files that belong to the project.

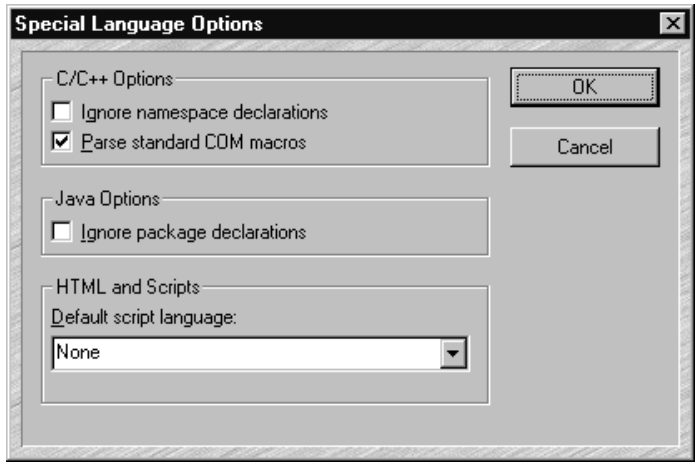
Global Conditions...

Click to edit the global condition set. Global conditions are defined for all projects. The total set of conditions defined for any given project is a combination of both the project-specific, and the global condition sets. The project-specific conditions will override global conditions with the same name.

See Also: "Edit Condition" on page 227.

Special Language Options

When you click the **Special** button in the Preferences: Languages dialog box, the Special Language Options dialog box appears. This dialog box controls special options for built-in languages.



C/C++ Options

Ignore namespace declarations

If checked, then namespace declarations are simply ignored in C++ code. All symbols declared within the namespaces are considered at the file scope, as though you did not write the namespace declaration.

If not checked (the default), then symbols declared within namespaces are considered in the namespace scope.

Parse standard COM macros

If checked, then the standard COM helper preprocessor macros, such as STDMETHOD, are recognized and parsed. Note that if you already have entries for these macros in your c.tom token macro file, then this option has no effect.

Java Options

Ignore package declarations

If checked, then package declarations in your Java files are ignored. Any symbols declared after the package statement are considered in the "global" package scope. That is, they are all in the same virtual package.

If not checked (the default), then any symbols declared after a package statement are considered in that package scope.

HTML and Scripts Options

Default script language

You can specify the default script language to use in HTML and ASP files with this control. The default script language is only used if another language is not specified in the script.

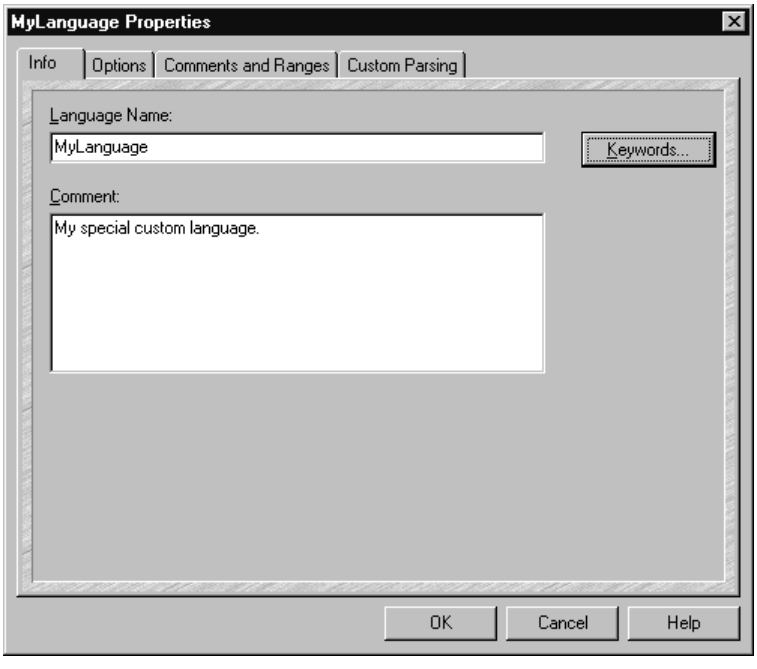
Language Properties

This command displays the properties of the currently selected language. The Language Properties dialog box appears when you click the Properties button in the Preferences: Language dialog box.

Source Insight supports two types of languages: Built-in and Custom. You can alter a few options for built-in languages. For custom languages, you can control all the parameters for a generic language.

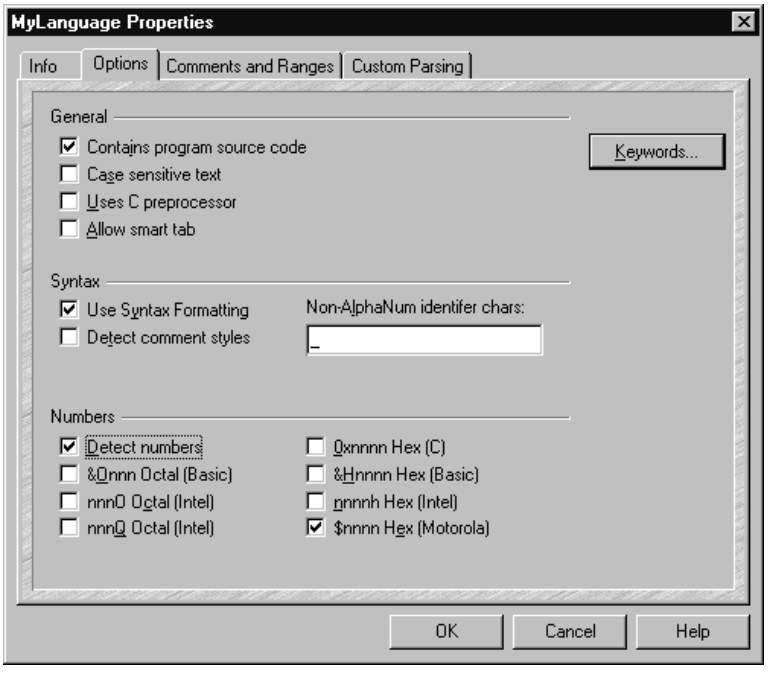
Language Info

The Info page is used to edit the name of the language, and a comment. Clicking the **Keywords...** button opens the Language Keywords dialog box.



Basic Language Options

Each language has basic options that govern how Source Insight treats files with this language. Built-in languages, such as C/C++ have fewer options in this page than do custom languages.



Contains program source code

If checked, then Source Insight will consider this a programming language. Certain features are altered when a programming language is used, as opposed to a simple textual language. For instance, references to declared symbols are displayed in the "Ref to …" styles.

Case sensitive text

This indicates whether the language is case sensitive or not. This affects how keywords are matched, as well as how symbols names are resolved in the symbol lookup engine.

Uses C preprocessor

If checked, then Source Insight will recognize #if and #ifdef preprocessor directives.

Allow smart tab

If checked, then the Smart Tab feature will be enabled when editing this type of language. If not checked, then Smart Tab will perform like a simple tab.

Use Syntax Formatting

If checked, then Syntax Formatting will be used when displaying files in this language.

Detect comment styles

If checked, then special comment styles will be detected. See "Comment Styles" on page 114.

Non-AlphaNum identifier chars

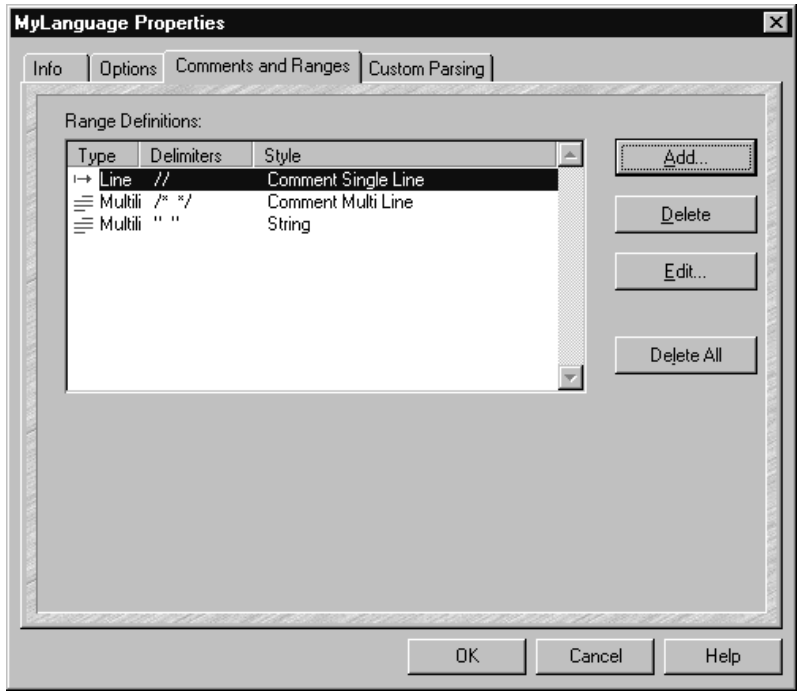
This text box contains the set of all valid non-alpha-numeric identifier characters. Alpha-numeric strings are always considered identifiers.

Detect numbers

If checked, then numbers found in the text are formatted with the "Number" style. The check boxes following this enable special number formats for hex and octal numbers.

Comments and Ranges

The Comments and Ranges page is where you specify how comments and other multi-line range elements are parsed. A quoted string is an example of a non-comment multi-line range element.



Add...

Click this button to add a new range element. The Range Definition dialog box will appear.

Delete

Deletes the selected range element.

Edit...

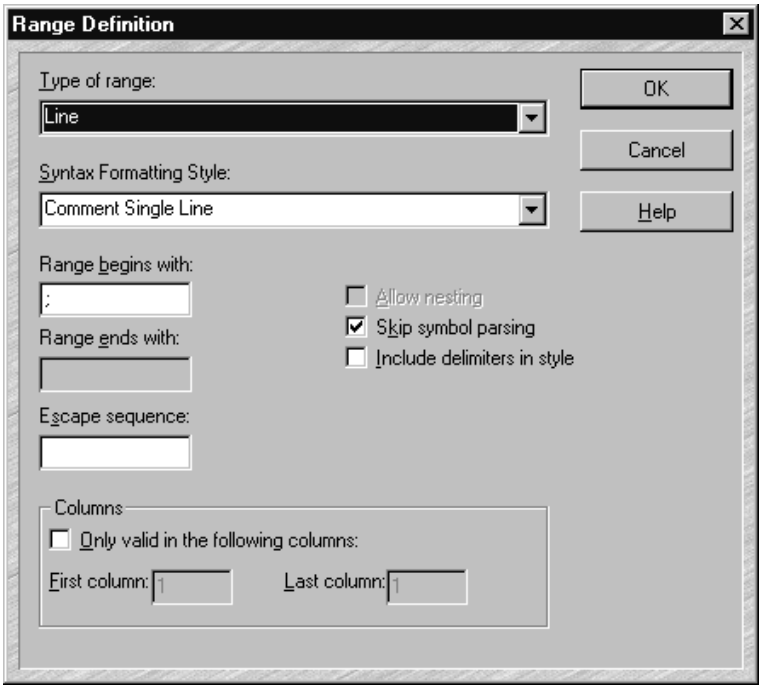
Opens the Range Definition dialog box so that you can edit the range element's properties.

Delete All

Deletes all range elements.

Range Definition

The Range Definition dialog box appears when you add a new range element, or edit a range element in the Language Properties: Comments and Ranges dialog box. It controls all the properties of a range.



Type of range

Select the type of range element from this list. There are two types of range elements:

1. **Line**. The range starts with a delimiter, and extends to the end of the line. It cannot span more than a single line.
2. **Multiline**. The range starts with a delimiter, and ends with another delimiter. The range can span more than single line, but it can also be contained within a single line. The starting and ending delimiter can be the same (such as a quote).

The list also contains presets for single and double quoted string ranges. When you select one of the presets, the parameters for the preset are loaded into the other text boxes in the dialog box.

Syntax Formatting Style

This specifies the syntax formatting style to apply to the range. Normally, you would select a comment style. However, you are free to select any style. If the range element describes a quoted string, you would probably want to select the "String" style.

Note: The style is applied to the whole range. The style overrides any other automatically applied style, such as "Reference To…" styles.

Range begins/ends with

These two text boxes specify the delimiter tokens that start and end the range. The tokens can be up to 15 characters long. If a Line range type is specified, then there is only one delimiter text box enabled. If you are specifying a Multiline range, the beginning and ending delimiters can be the same. This would be the case for a quoted string.

Escape sequence

If either the **Begin** or **End** delimiter is preceded by this escape sequence, then the delimiter is ignored. For example, you might specify backslash \ as an escape character in a quoted string so that you can embed quote characters inside the string like this: "a string with \"embedded\" quotes".

Allow nesting

This applies only if a Multiline range is specified. If this option is turned on, then the range can be nested. If the **Begin** and **End** delimiters are the same, nesting is not allowed because it doesn't make any sense.

Skip symbol parsing

The contents of the range will be ignored when the file is parsed for symbol definitions.

Include delimiters in style

The **Begin** and **End** delimiters are also formatted with the selected style. If this is unchecked, then the delimiters are formatted in the "Delimiter" style.

Columns

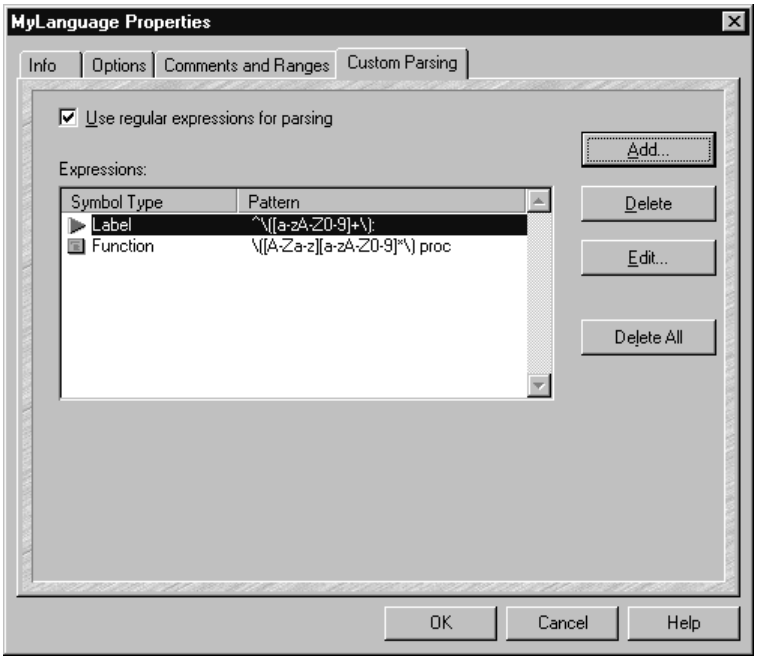
The columns group allows you to control if the range element should be sensitive to where on the line it occurs.

Only valid in the following columns

If enabled, then the range is only recognized if its **Begin** delimiter occurs in the range starting at the **First column** and up to and including the **Last column**. If this check box is unchecked, then the column is ignored.

Custom Parsing

The Custom Parsing page is where you can type a set of regular expressions to perform simple parsing operations on the language source files.



Use regular expressions for parsing

This enables the custom parsing expressions. Uncheck this to disable the use of custom parsing.

Expressions

This lists each parsing expression and the type of symbol it yields. When Source Insight parses a file, all the expressions are applied to the whole file.

Add...

Click this button to add a new parsing expression to the list

Delete

Click this to delete the selected expression.

Edit...

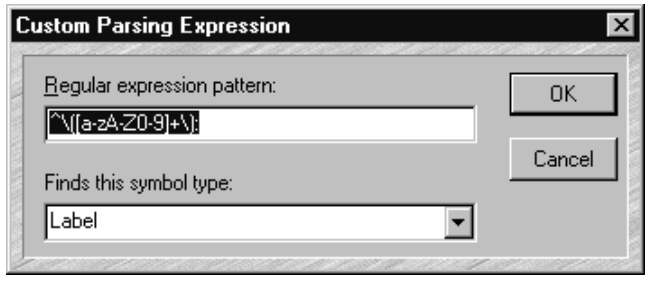
Click this to edit the expression.

Delete All

Click this to delete all the expressions from the list.

Custom Parsing Expression

When you click the **Add** or **Edit** button in the Custom Parsing dialog box, the Custom Parsing Expression dialog box appears.



Regular expression pattern:

This text box contains the regular expression used to parse a symbol definition out of a file. The expression should contain one group. The group describes what part of the matching pattern is the symbol name. Using a custom pattern allows you to parse symbols out of files for which Source Insight has no built-in knowledge. For example, the following string parses sections out of .INI files like WIN.INI.

```
^\[\(.*\)\]
```

For more information, see “Regular Expressions” on page 125.

Finds this symbol type:

This pull-down list specifies what type of symbol is found by the parsing pattern. The list contains all of the possible symbol types.

Note: When a symbol definition is parsed, the corresponding “Declare…” style is used when displaying the symbol name. For example, if the symbol definition is a Function, then the function name will be displayed in the “Declare Function” style.

Line Numbers

This command toggles the display of line numbers. The line numbers appear at the left of each line. They are displayed in the Line Number style. You can edit the style using the Style Properties command.

Link All Windows

The Link All Windows command toggles the Link All Windows mode. When the Link All Windows mode is on, then scrolling any window will scroll all the other windows. The Link All Windows mode overrides the normal window links that are setup by the Link Window command.

Link Window

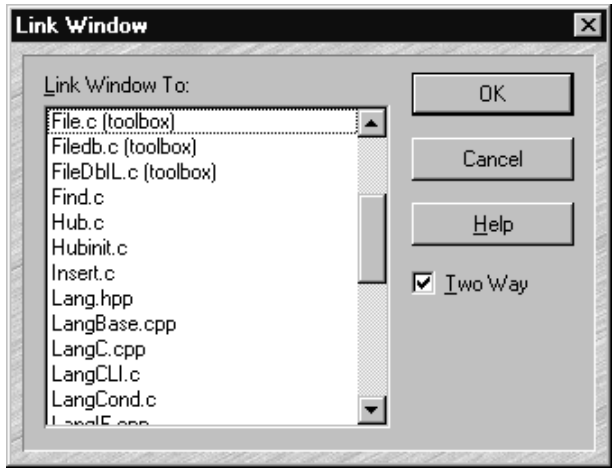
The Link Window command links the current window to another window so that they scroll together. This command lets you specify what window is linked.

Once you link the current window to another window, scrolling the current window will scroll the linked window.

If the **Two Way** check box is on, then the windows are linked together so that scrolling either of them will scroll the other. However, if the check box is off, then this is a one-way link. Scrolling the linked window will not scroll the other one.

Linkage	Result
Link A to B.	Window A has a link to window B. Scrolling window A will cause windows A and B to scroll, but not the other way around.
Link A to B to C	Windows may be linked to windows that are in turn linked to other windows. Scrolling window A, will cause all windows to scroll. Scrolling window B will scroll B and C.

Window links can even be circular. In other words, a window may be indirectly linked to itself. This is sometimes useful to get all the linked windows to scroll; regardless of what window you actually caused to scroll. Another way to get all windows to scroll is to use the Link All Windows command to turn on the Link All Windows mode, or just turn on the **Two Way** check box.



Line Window To

This list contains all windows, other than the current window. Select the window you want to link to here. Select <none> if you want to unlink the window.

Two Way

Turn this on to link both windows so that scrolling either of them will scroll the other. Turn this off to only perform a one-way link.

Load Configuration

The Load Configuration command allows you to load a new configuration file into the current configuration. You are able to load the whole configuration file, or just part of it.

Note: It is wise to keep a backup copy of your global configuration file, which will end up containing all your customizations. Once you use the Load Configuration command, or make a change to the customization settings inside Source Insight, the configuration file will be changed automatically.

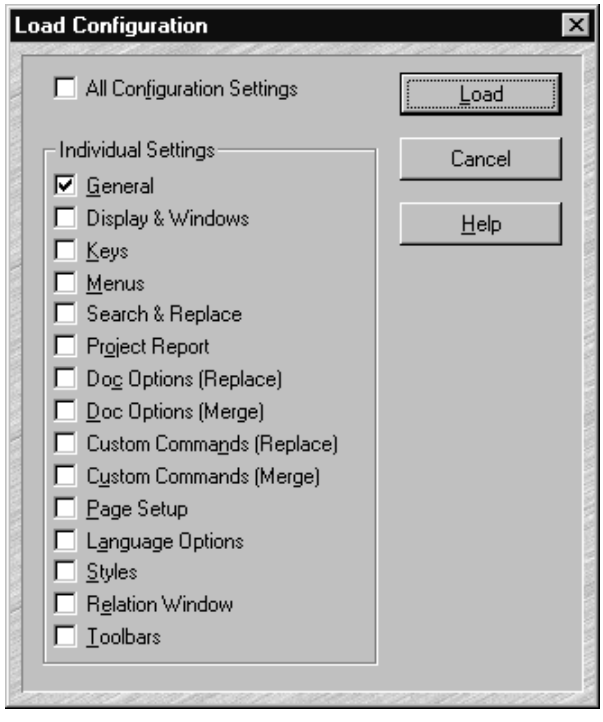
It is also a good idea to make a backup copy if you update your Source Insight software. Often, newer builds of Source Insight will be compatible with older configuration files, but not the other way around. If you should wish to revert to an older build of the software, it is best to use an older configuration file.

Global Configuration

Normally, Source Insight maintains a configuration file called *Global.cf3*, stored in the Source Insight program directory. You don't need to load or save this file yourself. That is done automatically. It is saved whenever you make any change to a configuration setting, and loaded when you open a project.

Partial Configurations

You can save a partial configuration file using the Save Configuration command. For instance, you may only save the key bindings. When you load a configuration file that contains a partial configuration, only the parts that exist in the file are loaded. The rest is unchanged. By loading and saving partial configurations, you can mix and match some of the configuration parts.



All Configuration Settings

Turn this on to load all parts of a configuration file. Turn this off to load only individual parts that are defined in the **Individual Settings** group below.

Individual Settings

Turning off the **All Configuration Settings** check box allows you to select the part of the configuration you want to load. For

example, this would allow you to load only the display settings, such as screen colors and screen size, while leaving other parts of the current configuration the same.

This grouping contains a check box for each configuration part. Check the items you want to load here. Configuration parts that consist of lists of named items, such as document types and custom commands, can either be loaded or merged.

Doc Options (Replace)

Check this to replace all the current document options with the document options in the configuration file.

Doc Options (Merge)

Check this to merge the document options in the configuration file with the existing document options. Merging means that the ones in the file being loaded replace document types with the same name, and document types that only exist in the file are added to the current list of document types already loaded. Any other existing document types are retained.

Custom Commands (Replace)

Check this to replace all the current custom commands with the custom commands in the configuration file.

Custom Commands (Merge)

Check this to merge the custom commands in the configuration file with the existing custom commands. Merging means that the ones in the file being loaded replace custom commands with the same name, and custom commands that only exist in the file are added to the current list of custom commands already loaded. Any other existing custom commands are preserved.

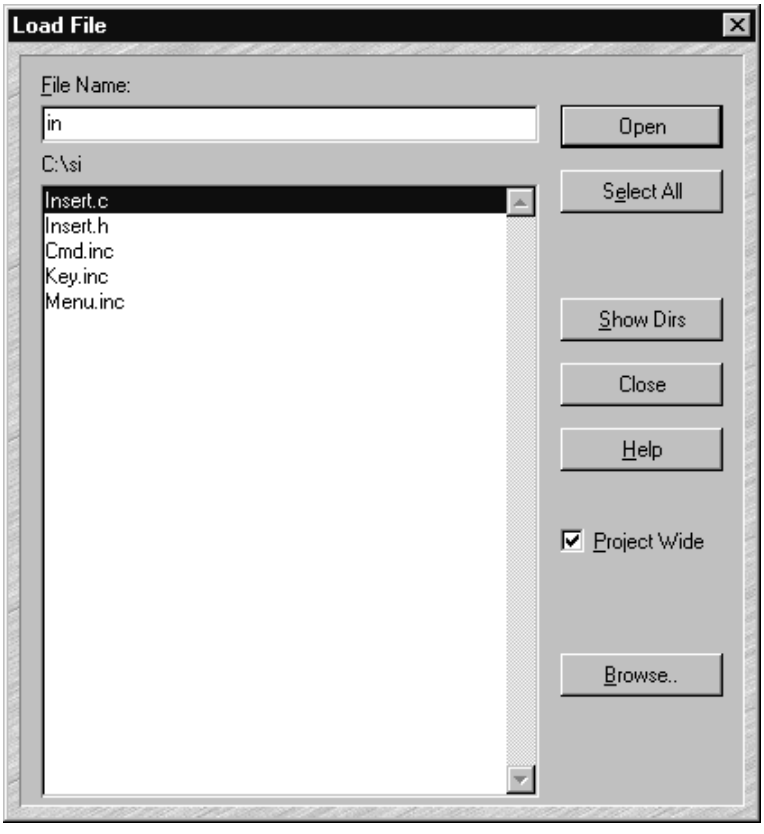
Load

Displays a File Open dialog box. You can select the configuration file you want to load with this dialog box.

Load File

The Load File command opens a file to edit. It opens a dialog box containing a list of all files in the project, regardless of directory.

You can also open files by using the Open command, or by dragging files from Windows Explorer and dropping them on the Source Insight window.



File Name

Type part of the name of the file you want to open. You may also type a series of wildcard specifications and click the **Open** button, and Source Insight will replace the **File list** contents with the results of wildcard expansion. The wildcards are expanded in the current directory.

As you type in this text box, partial matching occurs in the file list.

File list

If the **Project Wide** option is on, then this list displays all files in the current project, regardless of directory.

If the **Project Wide** option is off, then this list displays all the files in the current working directory. The current directory path is displayed above the list box. The files shown in the current directory are expanded from the wildcard strings specified in the Document Options dialog box for all document types.

Open

Click this to open the file selected in the file list. If nothing is currently selected in the file list, then Source Insight opens the file named in the **File Name** text box. If the **File Name** text box contains one or more wildcard specifications, then Source Insight will replace the file list with the results of the expansion. If the **Project Wide** option is on, the wildcards are expanded over the whole list of files in the current project; otherwise, the wildcards are expanded in the current directory. If the file specified does not exist, Source Insight will allow you to create a new file with that name.

Select All

Click this to select all the files listed in the file list.

Show Dirs

Click this button to toggle the list box contents between showing file names, and showing subdirectory names.

Project Wide

If checked, the file list will show all the files that have been added to the current project, regardless of directory. If not checked, the file list will show files in the current directory only. This option is always disabled if no project is open.

Browse

Click this button to bring up the standard system Open dialog box, which allows you to browse around your disks.

Opening a Project

If the file you specified has a .PR extension, Source Insight assumes it is a project file and will open that project. You can also drag and drop a .PR file onto the Source Insight application window to open the project.

Opening a Configuration File

If the file you specified has a .CF3 extension, Source Insight assumes it is a configuration file and will load that configuration into the current configuration. You can also drag and drop a .CF3 file onto the Source Insight application window to open the configuration. See Also: Configuration.

Opening a Workspace File

If the file you specified has a .WK3 extension, Source Insight assumes it is a workspace file and will load that workspace into the current workspace. You can also drag and drop a .WK3 file onto the

Source Insight application window to open the workspace. Opening a workspace will first close all existing open files. See Also: Workspace.

Loading a Clip File

If the file you specified has a .CLI extension, Source Insight assumes it is a clip file and will load that clip into the Clip Window. You can also drag and drop any text file onto the Clip Window to load it as a clip.

Load Search String

The Load Search String command loads the contents of the current selection into the current search pattern. The search pattern is what is in the Find text box of the Search and Replace commands.

Lock Context Window

This command locks the Context Window so that its contents don't change. When the Context Window is locked, it does not track your actions.

Lock Relation Window

Toggles Relation Window locking. When locked, it will not automatically update. You can still use the "Show Relation" command to manually update the Relation Window. You can also click on the Refresh Relation Window button in the Relation Window toolbar.

Lookup References

The Lookup References command searches the current project for references to a selected symbol. For example, click inside "BeginPaint", run the Lookup References command, and Source Insight will open a Search Results window, which lists all the places that reference "BeginPaint" in your project.

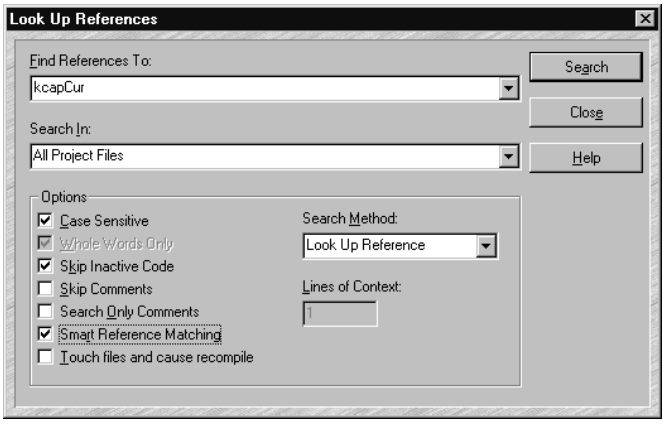
Source Insight uses its symbol indexes to make the searching fast.

References can be found in all source code text, including comments, and potentially inactive `#ifdef` branches. However, you can control whether these places are searched or not.

Search Project is the same command, with different option state.

The Lookup References command is very similar to the Search Project command. In fact, each dialog box is identical. However, each dialog box has its own persistent state.

Lookup References Dialog box



Find References To

Typing the symbol name you want to locate. The word under the cursor is automatically loaded into this text box. Source Insight will use the context of the cursor position to determine the exact symbol instance you want. If you invoked Lookup References from a symbol dialog box or window, then Source Insight keeps the exact symbol references along with this text box.

Typically, you would typing the name of an identifier in your program, however you can type any string here and a project-wide search will be performed. The search is very fast if you typing a single word only.

Search In

This drop-down list contains a list of document types. You can use this list to restrict the search to only a particular type of file, or just the current file. If the Project Window is visible, then you can also use this list to specify the files selected in the Project Window.

Search Method

You can pick the search method to use from this list. There are four different searching methods available:

- Simple String.
- Regular Expression – interprets the pattern as a regular expression.
- Keyword Expression – similar to an Internet search query.
- Lookup Reference – searches for symbol references.

Lines of Context

This only applies if you selected the Keyword Expression search method. This specifies how closely, in number of lines, the keywords must occur in order to qualify as a match. See Keyword Searches below.

Find word variations

If enabled, Source Insight will also find different ending forms of the keywords you specified. For example, if you specified the keyword "open", Source Insight will also find "opens", or "opened", or "opening". This option is only available for the Keyword Expression search method.

Options

Case Sensitive

Specifies whether the search is case sensitive or not.

Whole Words Only

For the Lookup References mode, this option is always on. If you choose a different search method, this will restrict matches to only whole words.

Skip Inactive Code

If enabled, then only code that is active under conditional compilation is searched. You must first specify known conditions in the Preferences: Languages dialog box, in order for Source Insight to know what conditions are active or not. Conditional compilation only applies to some languages.

Skip Comments

If enabled, then comments will not be searched.

Search Only Comments

If enabled, then *only* comments will be searched. This is mutually exclusive with the **Skip Comments** option. The comment options slow the search down a little.

Smart Reference Matching

This enables Source Insight's smart reference matching feature. Source Insight will determine whether each reference found is actually referring to the symbol you are looking for.

Matching exact references slows the reference finding process.

The **Smart Reference Matching** option means that the search results will only contain references strictly to the exact symbol you specified. For example, if you select a member of a struct and look up its references, the search results will only contain references to that particular member of that particular struct – not just any string that is equivalent. Note that this option slows the process down because each same-string occurrence has to be qualified with a symbol lookup.

Touch files and cause recompile. Turn this on to cause each file's "last modified" time stamp to be set to the current time. This is useful if you have a compile time dependency on an identifier usage. Just turn this on and search for references with this command. The places where the identifier is referenced will be "touched" and your make program or development system will recompile those files the next time you build your program.

Keyword Expressions

A keyword expression search is similar to an Internet search engine query. Source Insight searches the project for occurrences of a set of keywords that appear within a specified number of lines. The **Lines of Context** text box indicates the maximum distance the keyword terms can be from each other to qualify as a match.

For example, if you typed "cat food", then Source Insight will search for occurrences of "cat" and "food" within X lines of each other.

There is an implicit logical-AND operator between keywords. That is, if you type more than one keyword, the both keywords must be present to qualify as a match. You can include other Boolean operations as well. The following table lists the operators available:

Operator	Example	Action
AND or +	`cat and dog`	Both terms must be present.
OR	`cat or dog`	Either term must be present
NOT or – or !	`-cat`	The term must not be present
=	`=Betty`	Case sensitive match
? "regexp"	`? "^Ich"`	Term is a regular expression

You can also group expressions using parentheses. For example:

```
(cat or kitty) and food
(file or buffer) and (save or write) and !error
```

Keyword Variations

If you enabled the **Find word variations** option, then Source Insight will also find different ending forms of the keywords you specified. For example, if you specified the keyword "open", Source Insight will also find "opens", "opened", and "opening". This has the same effect as typing this expression:

```
(open or opens or opening)
```

Word variations are applied to each keyword term. For example, if you specified:

```
save write
```

Which implies "save" and "write" must be present. With word variations enabled, this search would be equivalent to:

```
(save or saves or saving) and (write or writes or writing)
```

Keyword Search Results

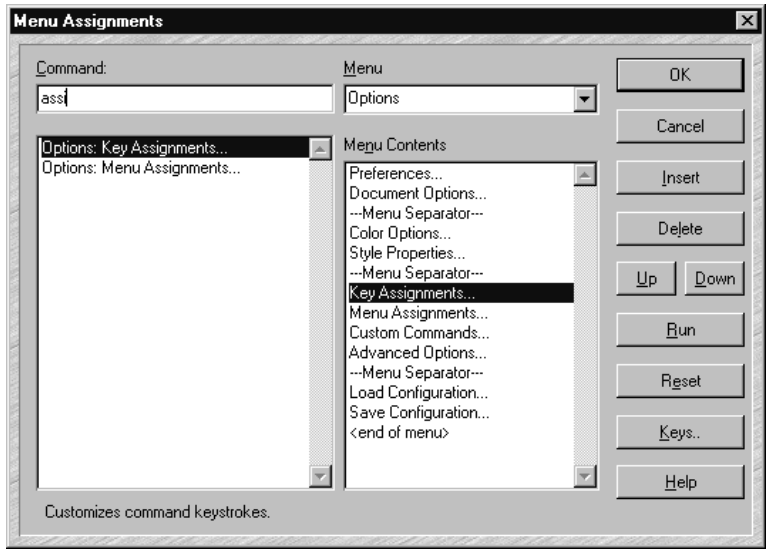
When you perform a keyword search, the Search Results will list blocks of lines that include the keywords together. This gives you a little bit of context around the matches.

Make Column Selection

The Make Column Selection is used with the mouse. Alt+Left Click and drag creates a rectangular selection. The column selection can be Cut, Copied, or Pasted.

Menu Assignments

The Menu Assignments command allows you to assign or reassign commands to the menus. The menu assignments are part of the current configuration.



Command

You can type into this text box to narrow down the command list, so that you can find the command you want easily.

Command list

This lists all the Source Insight commands, including macros and custom commands that you've defined. The commands are listed by category. When you select a command here, the menu it appears on, if any, is selected in the **Menu list**, and the contents of that menu are loaded into the **Menu Contents**.

Note that there is a special command in the list called "--Menu Separator--". Insert this item on the menu to create a separator line in the menu.

Menu list

This pull-down list contains the titles of all the menus. When you select a menu from here, the contents of the menu are loaded into the **Menu Contents**.

Menu Contents

This lists the menu items of the menu selected in the **Menu** list. Select a menu item here to indicate what item should be deleted, and where to insert new items.

OK

Click OK to record your menu assignment changes in the current configuration.

Cancel

Click Cancel to cancel the command. None of your changes up to this point will be saved in the current configuration.

Insert

Click this button to insert the selected command onto the selected menu. The command is inserted just before the selected menu item. You must select a command, a menu, and a menu item before clicking **Insert**.

Delete

Click this button to delete the selected menu item. You must select a menu item in **Menu Contents** before clicking **Delete**.

Up

Moves the selected menu item up, towards the top of the menu.

Down

Moves the selected menu item down, towards the bottom of the menu.

Run

Click this button to record your menu assignment changes in the current configuration, and run the selected command.

Reset

Click this to reset the menus assignments to their default, factory settings. Source Insight will ask you if you are sure you want to do this.

Keys...

Click this button to record the new menu assignments in the current configuration, and then switch to the Key Assignments command.

New

The New command creates a new, unsaved file buffer. The file is not created on disk until you save it using the Save or Save As commands.

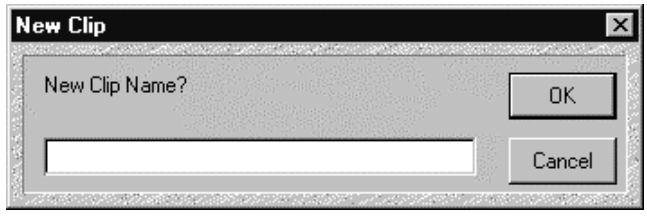
The New command will prompt you for a file name. By default, the new file is given a name of the form New0001.ext. The next time you use the New command, the new file name will be New0002.ext, and so on. The extension used will be the same as the current file. The name and extension you type will determine what document options will be in effect for that file.

When you save new files for the first time, Source Insight allows you to change their names, and asks if you want to add them to the current project.

Another way to create a new file is to use the Open command and specify a file name that does not exist. Source Insight will ask you if you want to create the file. Assuming you do, Source Insight will create a new unsaved file with the name you specified.

New Clip

The New Clip command creates a new clip buffer. You will be prompted for a clip name. You should not add extensions to the clip name.



New Relation Window

This creates a new Relation Window. You can have as many Relation Windows as you like. Each window has its own set of options.

For example, you could select a function name, and have one Relation Window showing what other functions it calls, and another Relation Window showing who calls the selected function.

Alternatively, you could have one Relation Window tracking the current selection, and another tracking the enclosing function.

New Project

The New Project command creates and opens a new project.

Since Source Insight allows only one project open at a time, Source Insight will ask you if it's okay to close the current project, if any, before proceeding. If you refuse to close the current project, the New Project command is canceled.

Where Should You Create A Project?

When you select a file in the New Project file dialog box, you are telling Source Insight where to store the project data files. This can be the same directory where your source files are, or you can pick a totally separate location.

If you creating a project for source files that are stored locally on your machine, there should not be any problem creating the project files in the same directory as your source files.

If you are creating a project that refers to files on a shared server, or any other place that you do not have permission to write to, then you should create the project somewhere on your local machine. You can use the Project Settings dialog box later to point the project root directory to the location of the source files.

The directory where you create the project will be the project's default root, or "home" directory. In the Project Settings dialog box, you can specify a different path for the project root directory. The project root directory typically is the path of the topmost directory containing your source files. When Source Insight displays file names, the files are displayed relative to project root directory. If you point the project root directory to the directory containing most of your source files, then you will not have to look at a lot of redundant path information.

In addition, the "Add new files automatically" feature (in Preferences: General) only will add new files automatically to your project if the files are in the project root directory or a descendent of this directory.

See Also: Synchronize Files, and Project Settings.

New Window

The New Window command opens a new window on the screen. The file appearing in the current window will appear in the new window as well.

Next File

The Next File command runs the Close command, and then runs the Open command. You will have an opportunity to save the current file if you've modified it, unless you have the Save Quietly option turned on in the Preferences: General dialog.

Next Relation Window View

Cycles through the view modes of the Relation Window. This includes the outline view, and graphical views.

Open

The Open command activates the Project Window and sets the focus on the text box of the Project Window. If the Project Window was not visible, it is made visible, and then hidden after you select a file to open.

You can customize what the Open command does in the Preferences: Files dialog box. Instead of activating the Project Window, you could have it bring up other dialog boxes or windows instead.

If you do not have a project open, then this command will bring up the system Open dialog box, which is a standard system dialog box.

You can also open virtually any type of file by dragging a file and dropping it onto the Source Insight application window. This includes project files (.PR), configuration files (.CF3), workspace files (.WK3), and clip files (.CLI), in addition to ordinary text files.

Opening a Project

If the file you specified has a .PR extension, Source Insight assumes it is a project file and will open that project. You can also drag and drop a .PR file onto the Source Insight application window to open the project.

Opening a Configuration File

If the file you specified has a .CF3 extension, Source Insight assumes it is a configuration file and will load that configuration into the current configuration. You can also drag and drop a .CF3 file onto the Source Insight application window to open the configuration. See Also: Configuration.

Opening a Workspace File

If the file you specified has a .WK3 extension, Source Insight assumes it is a workspace file and will load that workspace into the current workspace. You can also drag and drop a .WK3 file onto the Source Insight application window to open the workspace. Opening a workspace will first close all existing open files. See Also: Workspace.

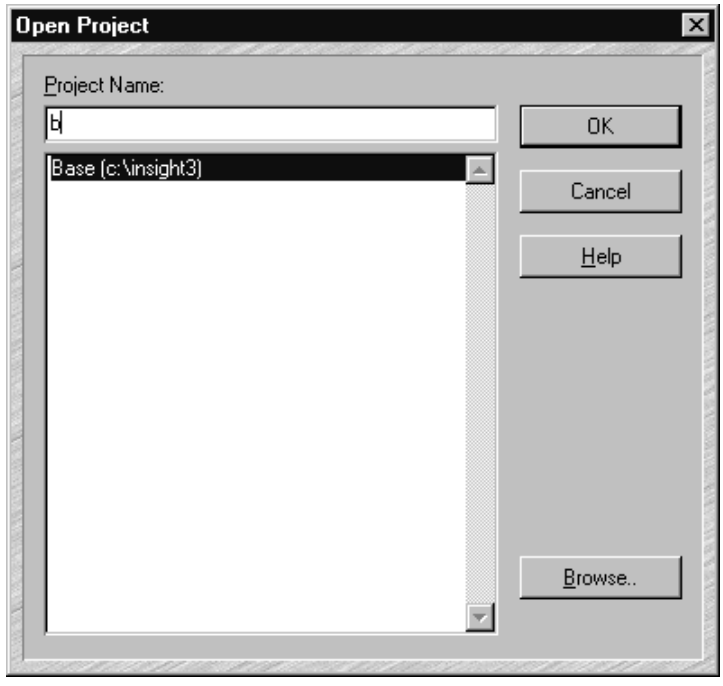
Loading a Clip File

If the file you specified has a .CLI extension, Source Insight assumes it is a clip file and will load that clip into the Clip Window. You can also drag and drop any text file onto the Clip Window to load it as a clip.

Open Project

The Open Project command allows you to open a new current project. It first closes the existing current project, if any, since Source Insight only allows one project open at a time.

You can also open a project by dragging a project file (.PR) from Windows Explorer onto the Source Insight window, or specifying a project file in the Open dialog.



Project Name

Type the name, or part of the name, of the project you would like to select.

Project list

Displays a list of all the projects you have created or opened on your machine. Select the project you want to open here. This list is simply a recording of known projects. You may have other projects that are not listed here. To open those, click the **Browse** button and locate the project file.

Browse

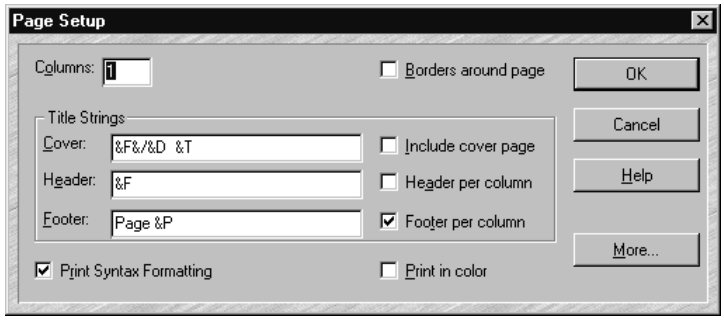
Click this button to bring up the standard Open dialog box, which allows you to browse around your disks. Locate a project file with a .PR extension. Once you select a file and close the Open dialog box, the file you selected will be loaded into the **Project Name** text box. Then click **OK** to open it.

Page Down

The Page Down command scrolls the active window down by one window full, with one line of continuity.

Page Setup

The Page Setup command allows you to control the layout of text on printed pages that are printed with the Print command.



Columns

Specifies the number of columns to print per sheet of paper. Each column will get its own page number. For example, if you have two columns, then each sheet of paper will get two pages of source code printed on it. This is more useful if you print in landscape mode (where the paper is wider than tall).

Borders around page

If checked, then a single line border will be drawn around the text on the page.

More...

Click this button to open the standard system Page Setup dialog box. This dialog box allows you to set paper size, margins, and orientation (i.e. Portrait vs. Landscape). You can also change the current printer settings from this dialog box.

Title Strings group

The items in this group affect the titles printed on the cover page, the top of the page (header), and the bottom of the page (footer).

Cover, Header, Footer

These text boxes specify the formatting codes to be used for the cover page, and the header and footer on each page. See below for details on the format of the strings.

Include cover page

If checked, then a single cover page will be printed before all other pages. Typically, you would put your name in the **Cover** string

text box so that someone at the printer could identify the source of your print job. If not checked, then no cover page is printed.

Header per column

If checked, then a header will be printed above each column. This only has an effect if the number of columns is set to two or more. If not checked, then a single header is printed at the top of the page.

Footer per column

If checked, then a footer will be printed below each column. This only has an effect if the number of columns is set to two or more. If not checked, then a single footer is printed at the bottom of the page.

Formatting Options

The formatting options control how syntax formatting is output to the printer. These options work independently of the screen settings, which you control in the Preferences: Syntax Formatting dialog box.

Print Syntax Formatting

If enabled, all syntax formatting effects will be printed, as well as displayed on the screen.

Print in color

If enabled, then color is used for printing. If your printer is not a color printer, then colors will be displayed in shades of gray. If you disable this option, then all text, regardless of on-screen color, is printed in full black or white. However, text that is colored near a neutral gray is printed in gray.

Header and Footer Codes

A *header* is a title printed at the top of the page. A *footer* is a title printed at the bottom of the page.

You can customize headers and footers for printed pages with the Page Setup command. You do this by using the following codes in the **Header** and **Footer** text boxes:

Code	Result
&L	Left-align characters that follow. This is the default.
&C	Center characters that follow.
&R	Right-align characters that follow.
&/	Move to the next line.
&B	Print the characters that follow in Bold.
&I	Print the characters that follow in Italic.
&U	Print the characters that follow in Underline style.
&D	Print the current Date.
&T	Print the current Time.
&F	Print the File name.
&P	Print the Page number.
&N	Print the total Number (N) of pages in the document. For example, in a file 12 pages in length, you would type Page &P of &N to have Page 1 of 12, Page 2 of 12, Page 3 of 12, etc., printed on the pages.
&&	Print a single ampersand (&) as a literal character rather than as an instruction.

Example:

To print "Confidential" at the left margin, center the page number, and print current date at the right margin, type:

```
&LConfidential&C&P&R&D
```

Page Up

The Page Up command scrolls the active window up by one window full, with one line of continuity.

Paren Left

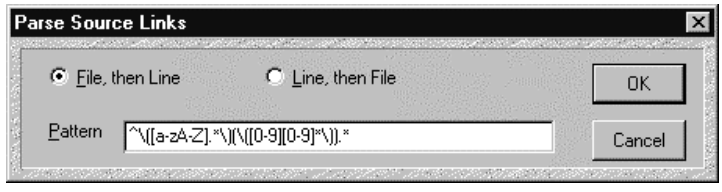
The Paren Left command moves the insertion point left to the next enclosing parentheses.

Paren Right

The Paren Right command moves the insertion point right to the next enclosing parentheses.

Parse Source Links

The Parse Source Links command searches the current file for a specified pattern. Whenever it finds a match, it creates a source link at that location. The source link links the source line in the current file to the file and line number that was parsed using the pattern.



File, then Line

and **Line, then File**. Select **File, then Line** if the first group in the pattern expression is the file name, and the second group is the line number. With this setting, the second group, (i.e. the line number), is optional.

Select **Line, then File** if the first group in the pattern expression is the line number, and the second group is the file name.

Pattern

Contains the regular expression used to search the command output for file names and line numbers. This text box must contain a valid regular expression that contains "groups" for the file name and the line number. The contents of this text box are saved in the current workspace.

The Parse Source Links command is useful if you have some kind of log file that contains compiler output and error messages. You just open the log file, and run the Parse Source Links command. A link will be setup for each line in the log file containing an error message. See Source Links for more details.

Maintaining Multiple Parse Patterns

If you often have more than one type of parse pattern that you want to use, you can use Custom Commands to define a custom command for each type that simply echoes the file and parses source links from the output. Each custom command can have its own parse pattern, so you can save many parse patterns this way.

For example,

1. From the Options menu, select Custom Commands.
2. Type "`Parse Type 1`" in the **Name** text box.
3. In the **Run** text box, typing "`command /c type %f`". This command line will type out the contents of the current source file.
4. Turn on the **Parse Source Links** option.
5. Type the parse pattern you want in the **Pattern** text box.

When you run this command, the current file will be typed out, captured by Source Insight, and parsed for source links using the parse pattern stored with the custom command.

Paste

The Paste command copies the contents of the clipboard to the current selection. If the current selection is already extended, it is deleted before pasting.

Paste From Clip

The Paste From Clip command copies the contents of a clip buffer to the current selection. The Clip Window is activated when you use this command so that you can specify the source clip name.

Double-clicking on a clip in the Clip Window also runs this command.

Paste Line

The Paste Line command moves the insertion point to the beginning of the current line and executes the Paste command. Cut Line, Copy Line, and Paste Line can be used together to quickly move whole lines around in a file.

Paste Symbol

(On the Symbol Window right-click menu)

The Paste Symbol command pastes the text in the Clipboard just before the selected symbol in the Symbol Window.

Play Recording

The Play Recording command plays back a command recording. Commands are recorded by using the Start Recording command. If the recorder is on when you use the Play Recording command, then recorder is automatically turned off first.

Preferences

Lets you specify user options. This one, multi-page dialog box contains several tabs for various types of options, such as Display, Files, and Syntax Formatting.

For more specific information, see:

See "General Options" on page 234.

See "Typing Options" on page 364.

See "File Options" on page 231.

See "Language Options" on page 254.

See "Symbol Lookup Options" on page 351.

See "Display Options" on page 211.

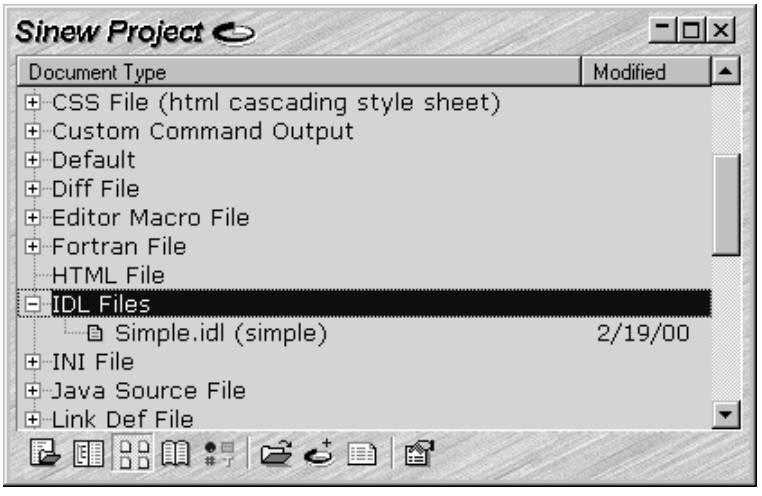
See "Color Options" on page 193.

See "Syntax Decorations" on page 356.

See "Searching Options" on page 333.

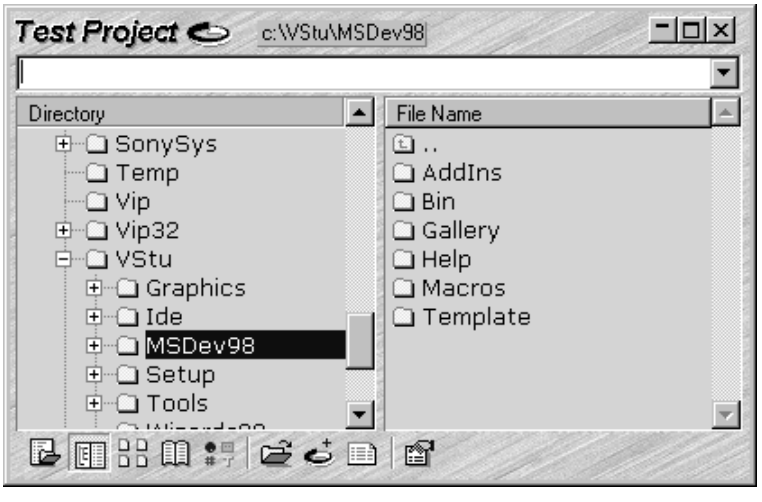
Project Document Types

Displays a list of project files, categorized by document type, in the Project Window.



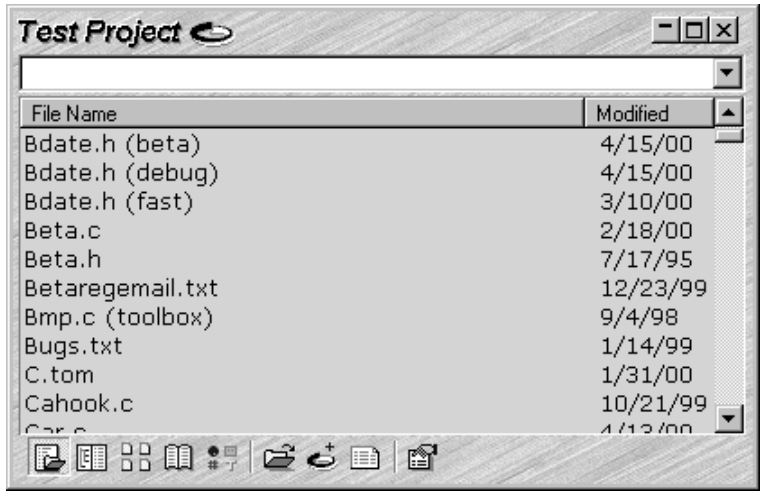
Project File Browser

Displays the File Browser view in the Project Window. This view allows you to browse around your disk.



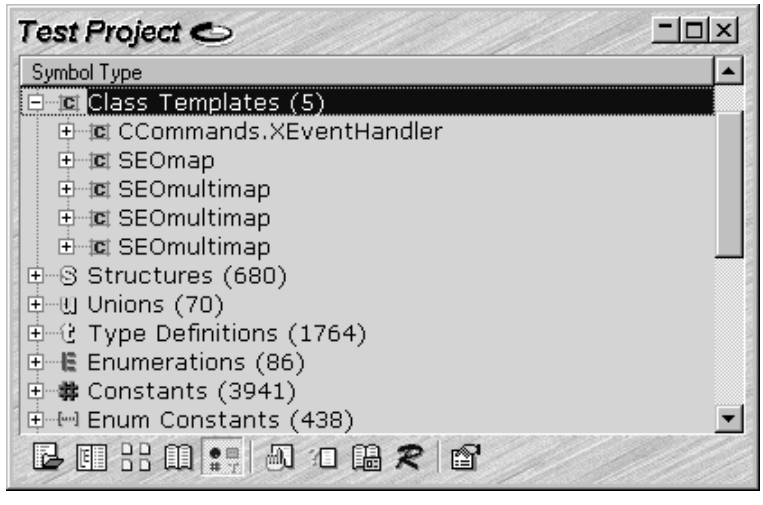
Project File List

Displays all project files in the Project Window. This is a "flattened" list of all files in the current project, regardless of directory.



Project Symbol Classes

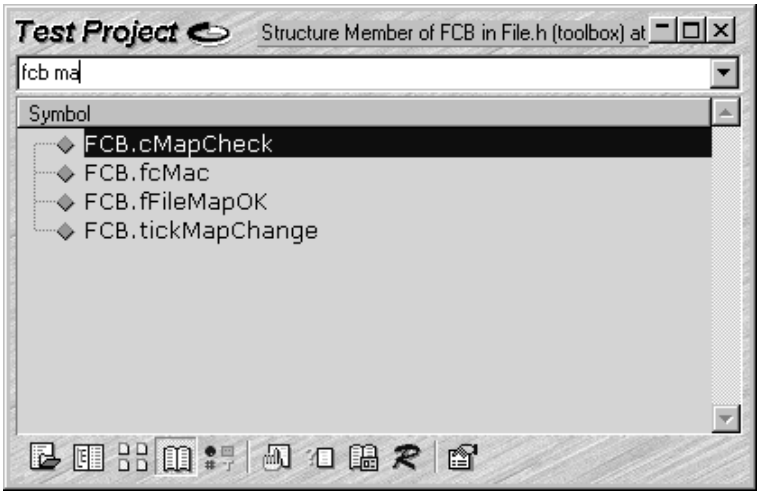
Displays project symbols by category in the Project Window.



Project Symbol List

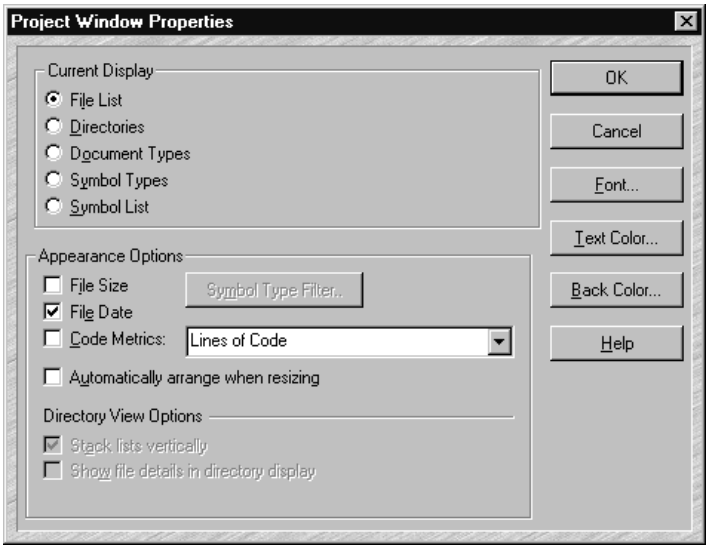
Displays all project symbols in the Project Window. You may find this view more useful than the Browse Project Symbols command

(F7). Unlike the modal dialog box, the Project Window is modeless, and the Context Window and the Relation Windows will slave to it.



Project Window Properties

Edits the properties of the Project Window.



Current Display

Select the view mode of the Project Window. Another way to select the view mode is to type Ctrl+Tab into the Project Window. You can also use the toolbar buttons to select the view mode.

Appearance Options

The appearance options control the visual content of the Project Window.

File Size, File Date

For view modes that show files, enable these to show the size, and/or date of each file in separate columns.

Code Metrics:

Show the selected code metrics column. The drop-down list to the right selects which code metric value to display in the column.

Symbol Type Filter

Click this button to choose the types of symbols to display in the symbol lists that are displayed in the Project Window.

Automatically arrange when resizing.

If enabled, the Project Window will automatically resize its columns when you resize the window. If disabled, then the columns widths are fixed, unless you change them.

Directory View Options

Stack list vertically

For the file browser view, this places the directory list above the file list. Disable this to show them side-by-side. If the **Automatically arrange when resizing** option is enabled, then the Project Window will switch the orientation automatically depending on the aspect ratio of the Project Window.

Show file details in directory display.

This is used in the File Directory view only. If enabled, then file details, such as the file size, are displayed in the file list.

Font, Text Color, Back Color

Click on these to select the font, text color, and background color, respectively.

Print

The Print command prints the current file. The standard Windows Print dialog box will appear.

You can control what font is used for printing files with the Document Options command. The Document Options command

lets you specify what font should be used for printing and what font should be used for the screen. Alternatively, you can tell Source Insight to emulate the screen font when printing. If you have a TrueType font selected, then emulating that font on the printer should work just fine. If you have a raster font selected, such as "MS San Serif" or "Courier", then printing that font on the printer may look blocky, or the printer driver may just substitute that font with something similar.

Color Printing

If you want to print your files with syntax formatting and color, you need to make sure you have enabled those options in the Page Setup dialog box.

Print Relation Window

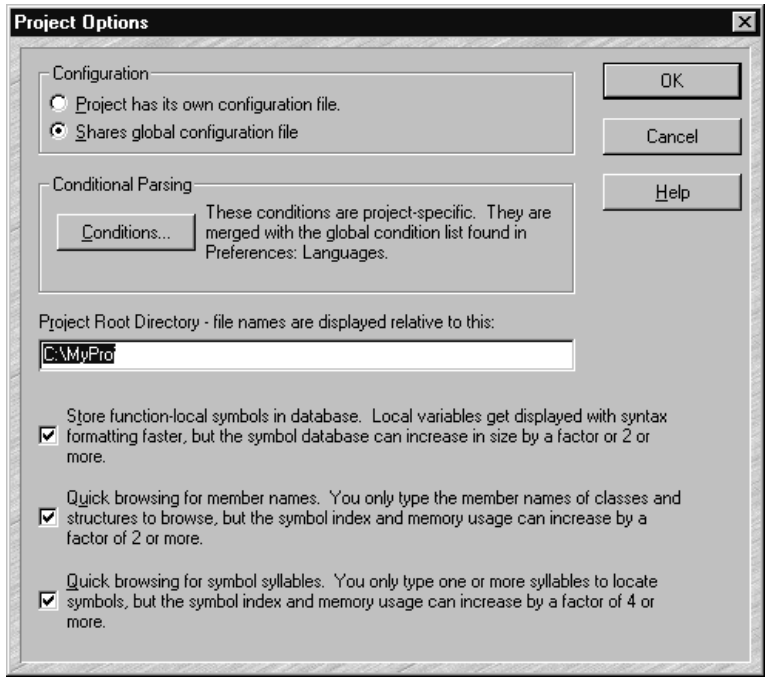
This command prints the graphic contents of the Relation Window. You can access this command by right-clicking on the Relation Window.

Source Insight can print a multiple page graph. Each page will indicate its page coordinate at the bottom. For example, if a graph spans a 4 x 3 page output, the bottom of the first page printed will contain "Cell 1,1 of 4 x 3 square".

Project Settings

The Project Settings command allows you to set various options that govern the current project. If no project is currently open, then the Project Settings command allows you to set the default options inherited by subsequently created new projects.

The Project Settings are saved with the project data file, in the project directory. These settings are independent of the configuration file.


Project Options
Configuration
Project has its own configuration file.
Shares global configuration file
OK
Cancel
Help
Conditional Parsing
Conditions...
These conditions are project-specific. They are merged with the global condition list found in Preferences: Languages.
Project Root Directory - file names are displayed relative to this:
C:\MyPro
Store function-local symbols in database. Local variables get displayed with syntax formatting faster, but the symbol database can increase in size by a factor or 2 or more.
Quick browsing for member names. You only type the member names of classes and structures to browse, but the symbol index and memory usage can increase by a factor of 2 or more.
Quick browsing for symbol syllables. You only type one or more syllables to locate symbols, but the symbol index and memory usage can increase by a factor of 4 or more.

Configuration Options

This specifies what configuration is used when the project is opened. The configuration is what holds most of your user customizations, such as key bindings.

Project has its own configuration file

This means the project has its own private configuration file, which is stored in the project directory. The name of the project configuration file is <project name>.CF3.

Shares global configuration file

The project shares the global configuration with other projects. The global configuration file is stored in the Source Insight program directory, and it is named *Global.cf3*.

Conditional Parsing

This controls the settings of condition compilation values.

Conditions...

Click to edit the conditions defined that are specific to the current project only. The conditions are only in effect when the current project is open, and only for files that belong to the project. Another

quick way to edit the conditions is to use the Edit Condition command.

The Project Root Directory

The *Project Root Directory* is the home directory of your source files.

This should contain the path of the directory that you want to consider the root, or "home" directory of the project. When Source Insight displays file names, it will show them relative to this directory. If you leave this text box blank, then Source Insight will use the directory where you created the main project file, with the .PR extension.

Letting you specify a different project root directory is useful if you want the project data files kept in a separate directory from the source files. The project data files are stored where you specified the .PR file, but the source files can be somewhere else. For example, you could create a project stored locally on your workstation, and add files from a remote network drive to the project. The file names will not contain extra path information if you specify the network drive path to the source files as the project root directory.

This feature is helpful when you are not allowed, or unable, to create project data files in the same directory as the source files. Some project administrators will allow you read-only access to the source code share point, and do not want you to put any Source Insight files there.

You can change the project root directory setting in the Project Settings dialog at any time after the project is created.

The project root directory path is stored with the project data in a relative format. The path is relative to the directory with the .PR file. That allows projects that were created on one machine to be opened remotely from another without confusing the logical drives. It also allows copying whole project directory trees to new locations.

Some Custom Command string meta-character substitutions are also affected by this path setting. The `%j` (project root directory) and `%v` (project root directory volume letter) refer to this path value, and not where the .PR file is.

Symbol Database Options

These options affect what is stored in the project's symbol database. You should choose these options *before* you add a bunch of files to your project. If you change these options after the project is already built, then the project will need to be rebuilt. Source Insight will rebuild it for you.

Store function-local symbols in the database

This will cause local variables, declared inside function bodies, to be stored in the symbol database. This will increase the database size, but syntax formatting for those variables will appear right away when you open the files.

Quick browsing for member names

If enabled, you only need to type the structure and class member names in order to perform partial matches on their names. However, the symbol index size and memory usage can increase by a factor of two or more. This option is recommended if you are using an object-oriented language primarily, so that you can find member functions and variables without having to type in the class name too.

Quick browsing for symbol syllables

If enabled, you only need to type one or two syllables of symbol names in order to perform partial matching on their names. However, the symbol index size and memory usage can increase by a factor of four or more. By indexing syllables, you can use syllable matching to quickly find symbols, even if you don't know what letters the symbol names begin with.

This option is not recommended if your project is very large, *and* you have a small amount of system RAM and/or swap space on your disk.

Index Performance

Syllable matching is such a useful feature, that we recommend enabling it, unless the above condition is true. If you think performance is affected, the symptoms of too large an index are:

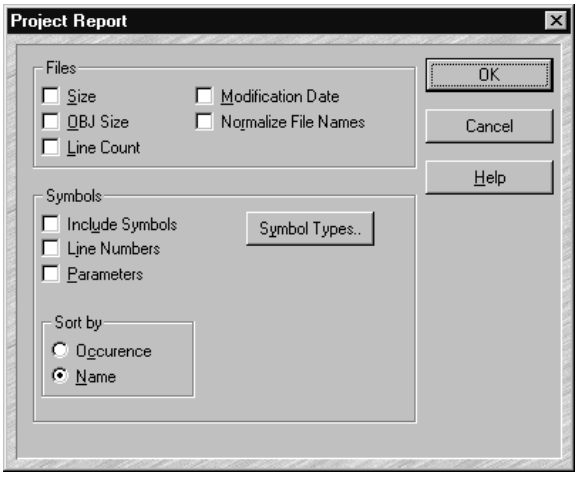
- Disk thrashing while building or rebuilding a large project, while little progress is being made.
- Opening or closing a project takes a long time.
- Synchronizing individual files is slow.
- Browse Project Symbols (F7) is slow to come up, accompanied by a lot of disk activity. Some delay is normal the first time you use it.
- Your project has over 2 or 3 million index entries. Obviously, this limit depends on the amount of RAM you have.
- Your hard drive light never seems to go off, or your system pauses for a long time while the disk is flushed.

If you experience slow-downs and you have a large project, (say over 200,000 symbols,) you should try turning off **Quick browsing for symbol syllables**. You can find out how large the database is by selecting Project > Rebuild Project and looking at the statistics on the bottom of the dialog box. Just cancel this dialog box when you are done. If the index entries are in the millions, then things can start to slow down. However, a Pentium III class machine with 256 MB of RAM should handle this size project well.

Adding more memory to your machine will improve performance.

Project Report

The Project Report command generates an output report file called <project>.RPT, which contains a list of files and symbols in the current project.



Files

Turn on these check boxes to include the corresponding information.

Include Symbols

If checked, then symbols will be listed under each file. If not checked, then no symbol information will be listed in the project report.

Line Numbers

If checked, then the line number where the symbol is defined is listed by the symbol name.

Sort by

If **Occurrence** is selected, then symbols will by listed by line number. If **Name** is selected, then symbols will be listed by symbol name.

Symbol Types

Click this button to indicate what types of symbols will be included.

Project Window command

The Project Window is a floating/dockable window that displays a list of the files in the current project. Double clicking on an entry in the Project Window list opens the file. Dragging a file onto the Project Window from Explorer or File Manager adds the file to the project. There is also a toolbar at the bottom of the Project Window for other commands relating to projects.

The Project Window can be toggled on and off by running the Project Window command. You can activate and set the focus to the Project Window by running the Activate Project Window command. This command will make the Project Window visible if it was hidden.

If a project is open, then the Open command activates the Project Window.

If the Context Window is open, then it will display the contents of any files selected in the Project Window.

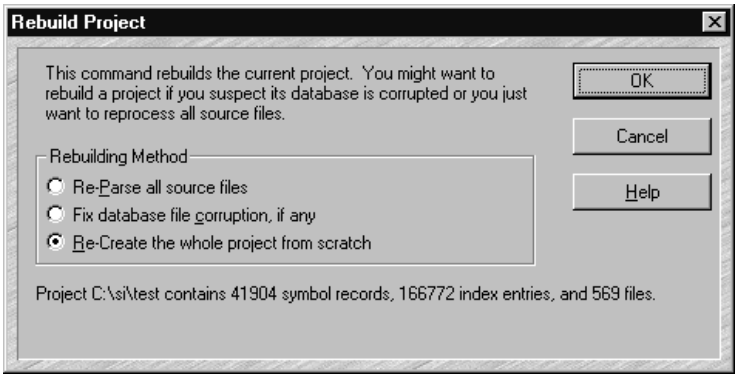
Rebuild Project

The Rebuild Project command rebuilds the project database files.

You may want to rebuild the project to get all the files re-parsed after a large change, or if you suspect the project data is not correct.

A project may become corrupted if Source Insight was abnormally aborted without closing the project.

The Rebuild Project dialog box also lists some statistics about your project. The number of symbol database records, symbol index entries, and files is displayed. This information is also output by the Project Report command.


Rebuild Project
This command rebuilds the current project. You might want to rebuild a project if you suspect its database is corrupted or you just want to reprocess all source files.
OK
Cancel
Help
Rebuilding Method
Re-Parse all source files
Fix database file corruption, if any
Re-Create the whole project from scratch
Project C:\si\test contains 41904 symbol records, 166772 index entries, and 569 files.

There are three methods of rebuilding the project. The last method is recommended.

Re-Parse all source files

This simply scans all the files in the project, and re-parses them to update the symbol database. This is the slowest method. However, if you cancel the operation, the symbol database will be left intact, although it may be slightly out of date. If you cancel, you can continue to edit normally and the re-parsing will continue in the background. The symbol database will be at least as current as it was when you began the rebuild process.

Fix database file corruption, if any

This option scans the databases and ensures that the databases are in a legal, self-consistent state. However, some files or symbols may be missing from the databases after the rebuild is complete if you recently added or removed files from your project. This is the fast method. You should use this command if you suspect that your project is corrupted. Normally, Source Insight can detect if a project was not closed properly when it tries to open the project. If Source Insight detects that the project is corrupt, it will ask you if you want to rebuild the project. Source Insight will not knowingly allow you to open a corrupted project.

Re-Create the whole project from scratch (recommended)

This is the most thorough method of rebuilding the project, and it is the recommended method. This deletes all symbolic information about the project, and rescans all the source files to regenerate the symbol database from scratch. It may take more time than the **Fix database file corruption** method. If you cancel it, the symbol database will be only partly complete. Source Insight will only have knowledge of symbols in the files that have been scanned. The background synchronization will complete the parsing. Alternatively, you can complete the process by using the Project > Synchronize Files command.

Record New Default Properties

(On the Symbol Window right-click menu.)

This makes the current settings of the window become the new Default settings for new windows.

For the Symbol Window, this records the window's width, symbol sorting, and symbol type filtering and uses those parameters as the new default for new windows created subsequently.

Redo

The Redo command reverses the effect of the Undo command. In effect, as you edit, Source Insight makes a list of the changes you've made to each file. The Undo command backs up through that list and the Redo command advances through the list.

You can also use the Redo All command to reverse all the Undo actions.

Source Insight keeps Undo/Redo history for each open file independently.

Redo All

Reverse the effect of all Undo actions. This brings the file all the way back up to date with your last change. It is equivalent to running the Redo command repeatedly until there is nothing left to Redo.

Redraw Screen

The Redraw Screen command redisplays the whole Source Insight screen.

Reform Paragraph

The Reform Paragraph command re-formats the selected paragraph of text so that all the lines in the paragraph are as wide as possible, within the margin width for the current document type.

If you have an insertion point selection, then the enclosing paragraph is reformed. If more than one paragraph is selected, then all the paragraphs in the selection are reformed.

If the first line of the paragraph is indented, then all subsequent lines in each paragraph will be indented by the same amount.

A paragraph of text is assumed a series of lines, bounded by blank lines.

You can specify the margin width in the Document Options command under the current file's document type.

For example, before running Reform Paragraph:

```
The quick brown
fox jumped
over the
lazy dog,
and other
mysterious sentences.
```

After running Reform Paragraph, the lines are made as wide a possible within the margin.

```
The quick brown fox jumped over the lazy dog, and other
mysterious sentences.
```

Refresh Relation Window

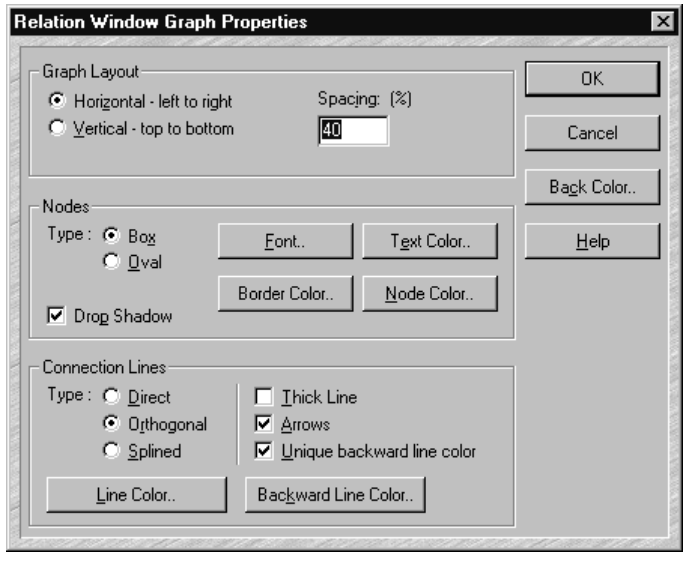
This command refreshes the Relation Window re-computing its contents.

Normally, the Relation Window updates automatically based on your selection.

If you prefer that the Relation Window not automatically update, you can either lock it with Lock Relation Window command, or use the Relation Window Properties command to set the "Automatic Symbol Tracking" to "None". Then you can use the Refresh Relation Window command to manually calculate the Relation Window contents.

Relation Graph Properties

Displays the graphing properties of the Relation Window. The graphing properties apply when the Relation Window is in a graph mode, not the outline mode.



Back Color

Click this to select a background color for the graph window.

Graph Layout

Selects the layout mode for the graph. You can choose either top to bottom, or left to right.

Spacing

Type a percentage that represents the scale factor of the inter-node graph spacing. A large percentage value will make the nodes spread out more.

Node Options

Node options control the appearance of node elements in the graph.

Type

Selects the shape of nodes.

Font

Selects the font used inside of each node.

Border

, Text, Back Color. Selects the colors used for the node

borders, text, and background fill.

Drop Shadow

Enable this to put a drop shadow on each node.

Connection Line Options

Connection Line options control the way the lines that connect nodes appear.

Type

Selects the style of lines that connect nodes.

- **Direct**. Lines are drawn straight between nodes.
- **Orthogonal**. Lines are drawn with right angle bends.
- **Splined**. Lines are drawn as curves.

Line Color

Selects the color used for connection lines.

Thick Line

Enable this to make connection lines thicker.

Arrows

Enable this to put arrowheads on the connection lines. If the **Spacing** percentage is small, the arrowheads are omitted due to the lack of inter-node space.

Unique backward line color

Enable this to have reverse flowing lines drawn in a separate color. Use the Backward Line Color button to select the color.

Backward Line Color

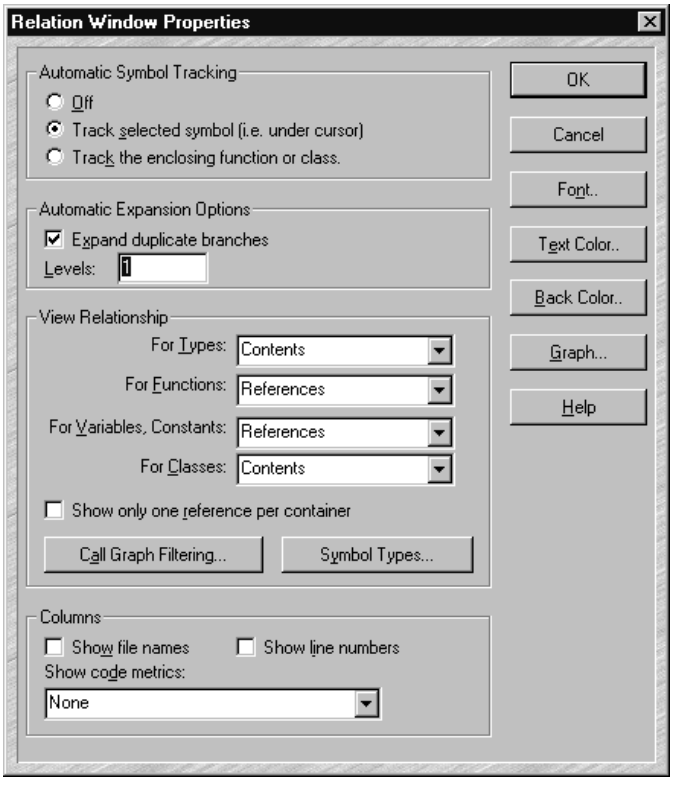
Selects the color used for reverse flowing lines.

Relation Window

Toggles the Relation Window visibility. This shows or hides *all* Relation Windows.

Relation Window Properties

The Relation Window Properties command is accessed from the Relation Window toolbar or shortcut menu. You control what relationships are shown from this dialog box, and how the window displayed.



Font..., Text Color..., Back Color...

Click on these to select the font, text color, and background color, respectively, of the Relation Window. These apply to the outline view only.

Graph...

Click on this button to open the Relation Graph Properties dialog box. From there, you can adjust the options for the graph view of the Relation Window.

Automatic Symbol Tracking

This set of options lets you select what the Relation Window tracks. It can track the symbol that appears under the cursor, or it can track

the enclosing function or data structure where the selection is located.

- **Off**. Nothing is tracked. The Relation Window is in manual mode.
- **Track selected symbol**. The Relation Window will describe whatever symbol is at the cursor position.
- **Track the enclosing function or class**. The Relation Window will track the function or class that contains your cursor. For example, if your cursor is inside the body of a function, that function becomes the subject of the Relation Window.

Automatic Expansion Options

This specifies how deep the Relation Window should expand automatically.

- **Levels**. The number of levels to expand below the root node.
- **Expand duplicate branches**. Enable this to expand duplicate child branches, even if they have already been expanded earlier. If this is disabled, then duplicate children are inserted in a collapsed state.

View Relationship Group

Relationship Rules

This group specifies the relationship expanding rules the Relation Window uses when it tracks symbols of different types.

The relationship shown depends on the type of symbol. You can specify what relationship is shown for different symbol types. For example, you could set the relationship viewed for functions to "Calls", and the relationship viewed for classes to "Inheritance". Thus, when you select a function, the Relation Window shows a call tree, and when you select a class name, the Relation Window shows the class inheritance hierarchy.

Each time the Relation Window expands a symbol to show a new level, the relationship represented by the expansion is based on the type of symbol being expanded. That means each Relation Window has the potential to show multiple relationships. For example, a class might show its contents, which consists of member functions. Each member function might show its references.

The "Type of" Relationship

There is a relationship in the list for Variables that is named "Type of". The "Type of" relationship yields the type of the variable. Then, the relationship rule for Types is applied. It is a kind of indirect relationship.

For example, let's say you set the Variable relationship to "Type of", and the Type relationship to "References". Now when you select a variable of a particular structure or class type, the Relation Window will decode the variable's type, then apply the "References" relationship rule and show the references to the variable's type.

Show only one reference per container

When the Relation Window is showing a "references" type of relationship, you can enable this to suppress duplicate references from inside the same object. For example, if function B is called three times from inside of function A, only the first reference from function A will be listed. If you disable this option, then all three references will be listed as three separate nodes.

Disabling this will provide more reference information. In the graph views, the multiple references are displayed nicely inside of a single graph node, making it easy to see how the references are concentrated among objects.

Columns

This group specifies what columns should appear in the outline list view. The Relation Window can be sorted by clicking on any column header.

Show file names

Shows the file name column.

Show line numbers

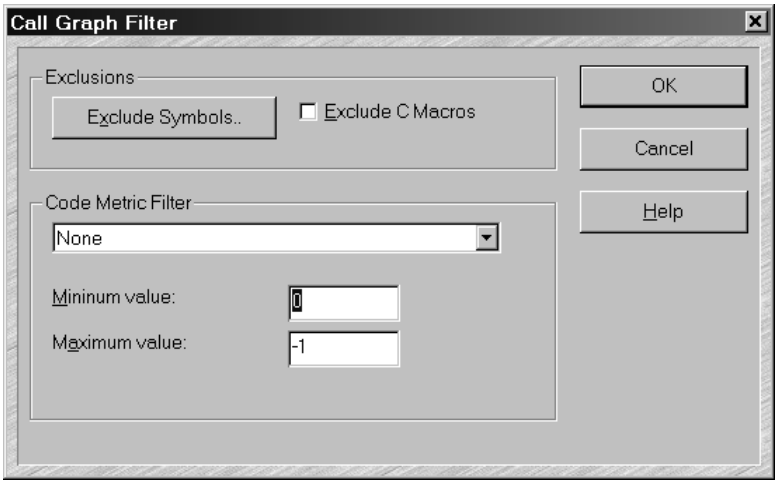
Shows the line number column. In most cases, the line number is where the given symbol is referred to.

Show code metrics

Shows the selected code metrics column. Only one code metrics column is allowed.

Call Graph Filter

The Call Graph Filtering dialog box allows you to control what symbols participate in the call tree calculation.



Exclusions

This section controls what symbols are filtered out of the call tree.

Exclude Symbols...

Click this button bring up the Exclude Call Graph Symbols list. You can add specific symbols to this list. If a symbol is in the exclusion list, then Source Insight will not expand the symbol in the call graph display.

Exclude C Macros

Check this to omit C function-like macros from the call graph. If this option is off, then function-like macros are expanded in the call graph as though they were an actual function.

Code Metric Filter

This section specifies the code metrics criteria for filtering out symbols. If enabled, symbols will only be included in the call graph is the symbols code metric value is within an acceptable range.

From the Code Metric Filter drop-down list, select the code metric that you want to use as the criteria, or select "None" if you don't want to use this option.

Minimum value

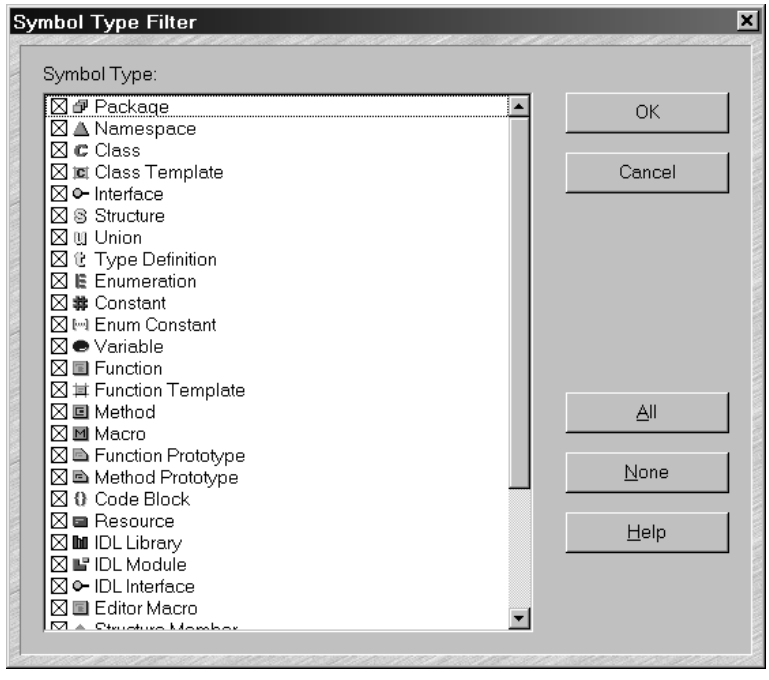
A symbol must have a code metric value of this or greater to be included.

Maximum value

A symbol must have a code metric value of this or less to be included. If the value is set to –1, then there is no maximum.

Symbol Type Filter

The Symbol Type Filter dialog box appears whenever you ask to specify symbol types to be used to filter an operation or listing.



Reload File

Reloads the current file from disk, losing *all* changes since saving. This has the same effect as closing the file without saving, and then opening the file again. The change history and undo history are also lost.

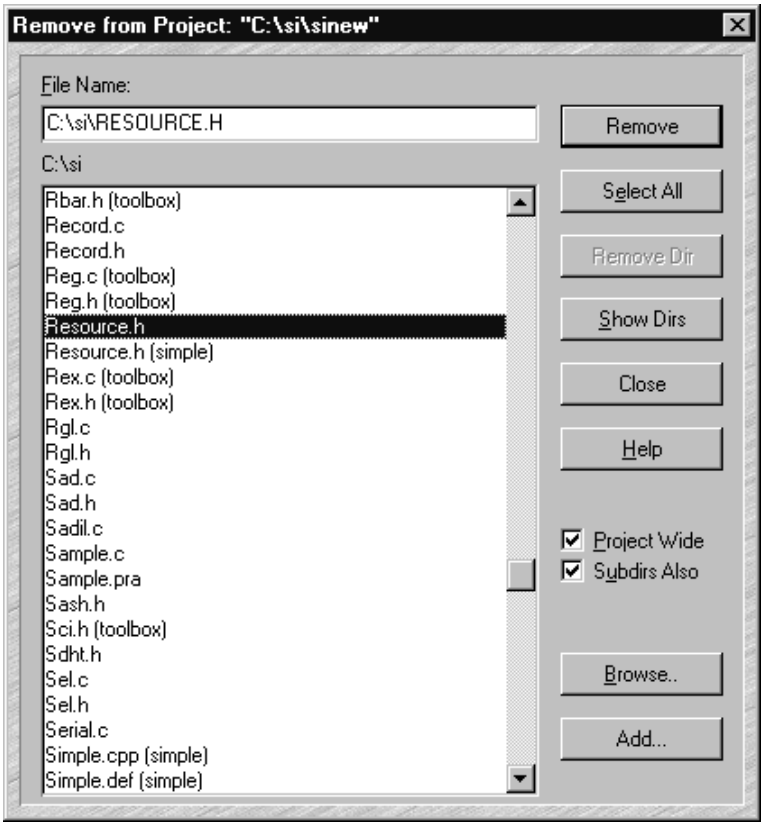
Reload Modified Files

This command will check the time stamps on each open file and reload those that have been changed outside of the editor. This only applies to files that were opened with the "Let other processes overwrite files" option turned on in the Preferences: Files dialog box.

You can have this work performed automatically in the background by selecting an option in the Preferences: Files dialog box.

Remove File

The Remove File command removes one or more source files from the current project. Note that the actual files are not deleted from the disk, but only removed from the project.



File Name

Contains the name of the file to be removed from the project. You can also add a wildcard and press Enter to expand the wildcard into the file list.

File List

Contains a list of all project files in the current working directory . If you select a file from this list box, the file name is loaded into the File Name text box.

Remove

Click this button to remove the selected file from the project. If the **File Name** text box contains wildcard characters, the wildcards

will be expanded and displayed in the File list box, and the dialog box will not be closed.

Select All

Click this button to select all the files contained in the file list.

Remove Dir

Click this button to remove the selected directory contents from the project. .

Show Dirs

Click this button to toggle the list contents between showing file names, and showing subdirectory names. When the subdirectories are shown, this button changes to "Show Files".

Browse

Click this button to bring up the standard Open dialog box, which allows you to browse around your disks.

Add

Click this to go to the Add Files dialog box.

Remove Project

The Remove Project command allows you to remove an existing project. When a project is removed, all project data files created by Source Insight are deleted. The Remove Project command will not delete your source files.

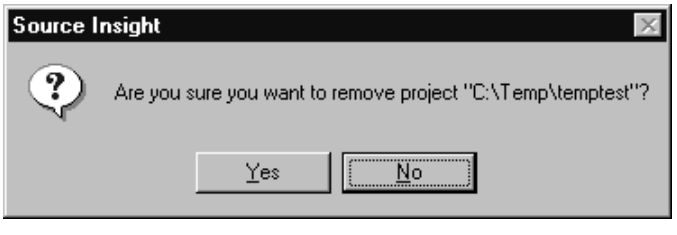
Project Name

Add the name of the project you want to remove.

Project list

Displays a list of all projects opened or created on your computer. Select the project you want to remove here.

Since the process of creating and adding source files to a large project can be somewhat time consuming, Source Insight confirms that you indeed want to remove the project.



Click **No** to cancel the Remove Project command. The project will not be removed. If the project you selected to be removed was previously open, then it will be closed at this point.

Click **Yes** to confirm that you want to remove the project.

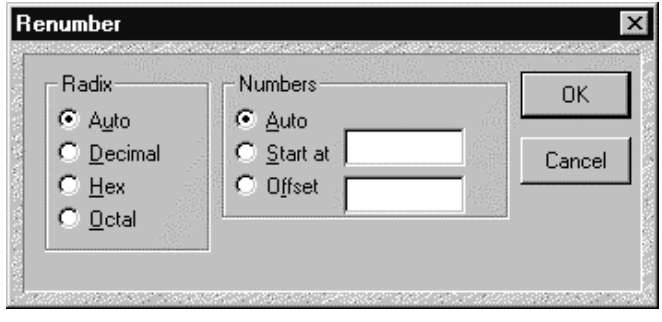
Rename

The Rename command renames the current file. The file does not have to be saved on disk. If the file is part of the current project, the project is adjusted to reflect the new name. You may also move the file to a new directory with this command, but you cannot move the file to a different drive.

The Rename command does **not** save the file.

Renumber

The Renumber command reorders numbers found in the current file, or just the current selection. If the selection is extended when the Renumber command is used, then only the selection is processed. If the current selection is an insertion point, then the whole file is processed. Renumber also works on a column selection.



Radix

Select a radio button in this group to specify the radix of the numbers generated by the Renumber command.

Auto

Use the radix of the original number, as determined by Source Insight.

Decimal

Use base 10.

Hex

Use base 16.

Octal

Use base 8.

Numbers

Specifies what action is to be performed on numbers found in the file. You can add values into the Start at and Offset text boxes in base 10, 8, or 16.

Auto

Replace numbers in ascending order, beginning with the value of the first number found.

Start at

Replace numbers in ascending order, beginning with this value.

Offset

Replace numbers with the same number, plus this value.

Radix Determination

Source Insight determines the radix of a number as follows.

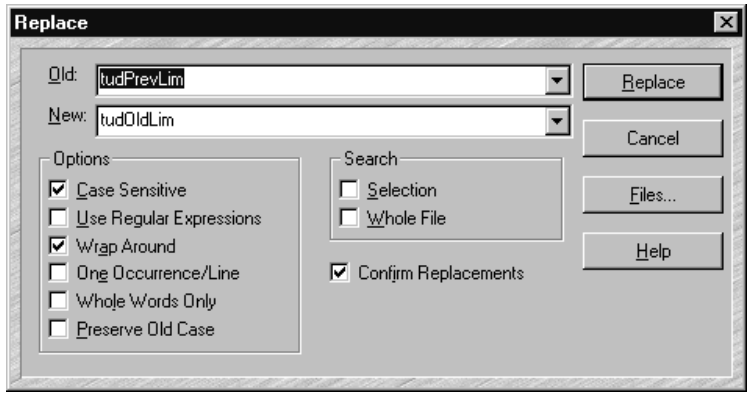
- If the number begins with `0x` then it is assumed hexadecimal.
- If the number begins with zero and a digit, it is assumed octal.
- Otherwise, the number is assumed decimal.

Repeat Typing

The Repeat Typing command repeats the last characters you typed. For example, if you select somewhere and type `abc`, then run Repeat Typing, another `abc` will be inserted automatically.

Replace

The Replace command searches for a specified pattern and replaces each occurrence with a new pattern. Only the current file is searched. The search can be done over the whole file, or just the current selection.



Old

Add the old pattern you want to replace in this text box. The pattern can be a regular expression.

New

Add the new pattern that should replace the old one in this text box.

Replace

Click this to begin the replacing operation.

Files

Click this button to transfer to the Replace Files command, where you can perform replacements in multiple files.

Options

Case Sensitive

If checked, Source Insight will only find matches if the case matches exactly.

Use Regular Expressions

If checked, the **Old** and **New** patterns are assumed to be regular expressions.

Wrap Around

If checked, the search continues at the beginning of the file when it reaches the end of the file. The search will wrap around only once. If not checked, the search stops when it reaches the end of the file.

One Occurrence / Line

If checked, only the first occurrence of the **Old** pattern on each line is replaced. If not checked, then *all* occurrences of the **Old** pattern on each line are replaced.

Whole Words Only

If checked, then Source Insight only finds matches that are whole words. If not checked, then Source Insight will also find matches that are embedded in words.

Preserve Old Case

If checked, then Source Insight will replace text but retain the upper and lower case of the original text. If not checked, then Source Insight will replace text using the case exactly as it appears in the **New** text box. This option is most useful when **Case Sensitive** is off.

This feature lets you replace all occurrences of a word, regardless of case, and still maintain the original case. For example, let's say you want to replace all "abc" and "ABC" with "xyz" and "XYZ" respectively. Add "abc" in the **Old** text box, add "xyz" in the **New** text box. Disable **Case Sensitive**, and enable **Preserve Old Case**.

Confirm Replacements

If checked, Source Insight will confirm each replacement by prompting you.

Search Group

The Search group of options specifies the scope of the search.

Selection

Searches only the currently selected text. This check box is automatically checked if the current selection is extended when the Replace command is invoked.

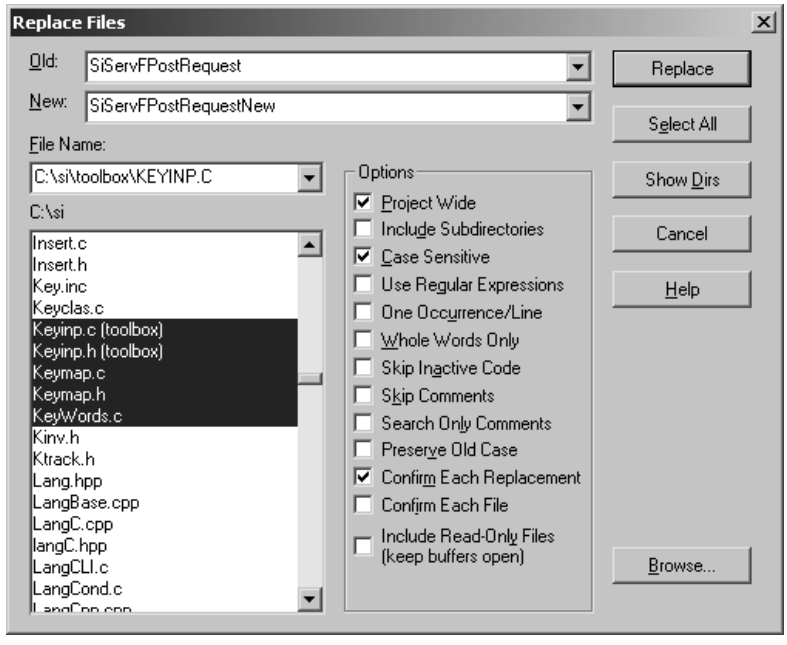
Whole File

Searches the whole file, from the first line to the last. This check box is automatically checked if the current selection is an insertion point when the Replace command is invoked.

Nothing checked in this group means to start searching at the current selection, and continue to the end of the file.

Replace Files

The Replace Files command searches for a specified pattern in multiple files and replaces each occurrence with a new pattern.



Replace

Click this button to begin the replace operation.

Select All

Click this to select all the files in the file list.

Show Dirs

Click this button to toggle the file list contents between showing file names, and showing only subdirectory names. When the subdirectories are shown, this button changes to "Show Files".

Old

Add the old pattern to be found and replaced in this text box. The pattern can be a regular expression.

New

Add the new pattern that should replace the old one in this text box.

File Name

The name of the file to search. You may also add a series of wildcard specifications and click the **Replace** button (or press Enter) and Source Insight will replace the file list with the results of the wildcard expansion. If the **Project Wide** option is on, the wildcards are expanded over the whole list of files in the current project; otherwise, the wildcards are expanded in the current directory.

If the **Project Wide** option is on, Source Insight will search the project symbol for file names added in the **File Name** text box, so you don't have to include a directory specification for those files.

File list

If the **Project Wide** option is on, then this list displays all files in the current project.

If the **Project Wide** option is off, then this list displays all the files in the current working directory. The current directory path is displayed above the file list. Source Insight shows only files for known document types in the current directory. The document types are specified with the Document Options command.

Options Group

Project Wide

This check box controls whether the File list shows all the files in the project, or just the files in the current working directory.

Include Subdirectories

If this check box is checked, then any selected directories are recursively searched. This option and the **Project Wide** option are mutually exclusive.

To recursively search a set of directories:

1. Uncheck the **Project Wide** check box.
2. Check the **Include Subdirectories** check box.
3. Select one or more directories in the file list.

You can also type a file wildcard specification in the **File Name** text box to limit the search to particular file extensions or names.

Case Sensitive

If checked, Source Insight will only find matches if the case matches exactly

Use Regular Expressions

If checked, the **Old** and **New** patterns are assumed to be regular expressions.

One Occurrence / Line

If checked, only the first occurrence of the **Old** pattern on each line is replaced. If not checked, then *all* occurrences of the **Old** pattern on each line are replaced.

Whole Words Only

If checked, then Source Insight only finds matches that are whole words. If not checked, then Source Insight will also find matches that are embedded in words.

Skip Inactive Code

If enabled, then only code that is active under conditional compilation is searched. You must first specify known conditions in the Preferences: Languages dialog box, in order for Source Insight to know what conditions are active or not. Conditional compilation only applies to some languages.

Skip Comments

If enabled, then comments will not be searched.

Search Only Comments

If enabled, then *only* comments will be searched. This is mutually exclusive with the **Skip Comments** option. The comment options slow the search down a little.

Preserve Old Case

If checked, then Source Insight will replace text but retain the upper and lower case of the original text. If not checked, then Source Insight will replace text using the case exactly as it appears in the **New** text box. This option is most useful when **Case Sensitive** is off.

This feature lets you replace all occurrences of a word, regardless of case, and still maintain the original case. For example, let's say you want to replace all "abc" and "ABC" with "xyz" and "XYZ" respectively. Add "abc" in the **Old** text box, add "xyz" in the **New** text box. Disable **Case Sensitive**, and enable **Preserve Old Case**.

Confirm Each Replacement

If checked, Source Insight will confirm each replacement by prompting you.

Confirm Each File

If checked, Source Insight will confirm each modified file by prompting you.

Include Read-Only Files (keep buffers open)

If checked, then replacements will be made inside of read-only file buffers. Source Insight will not attempt to save the file as the replacement operation progresses. The files will be left open and modified, allowing you to save the files yourself. If not checked, then read-only files will be skipped. Note that this options works independently from the Preferences: Files option **Allow editing read-only file buffers**.

You can make Source Insight automatically save over read-only files while replacing if you enable the Preferences: Files option: **Save over read-only files without prompting**.

Restore File

The Restore File command restores the current file to its *original* contents, as it was when it was *first opened*.

Reverting will lose all changes you made since it was first opened, even if you saved the file. The file that is saved on disk is not altered. Only the open file buffer is restored.

You should use caution with this command, since it effectively undoes any saving you performed on the file.

Note that you can use the Undo command to undo the Restore File operation.

Restore Lines

The Restore Lines command restores a block of edited lines back to their original contents. The lines will be as they were when you first opened the file.

Restoring lines will lose those changes you made since it was first opened, even if you saved the file. The file that is saved on disk is not altered. Only the lines in the open file buffer are restored.

The Restore Line command is undo-able. This gives you a powerful, out-of-order undo capability.

You can access the Restore Lines command quickly by right-clicking in the selection bar (left margin) area next to a block of modified lines and choosing it from the right-click menu. You can see what lines are modified by turning on line revision marks. (See Preferences: Display command.)

Save

The Save command saves the current file to disk. The file is saved to its current name. Prior to saving, any changes you made to the current file were only present in the unsaved version you were editing. The file on disk never is changed until you save the file by using the Save, Save As, or Save All commands.

A file can also be saved by answering "Yes" to the "Save changes to file?" message when you try to close a changed file.

If the file is new and has never been saved before, or the file is Read-Only, then the Save command runs the Save As command instead. The Save As command allows you to specify the name of the file to be saved.

Checkpoint File

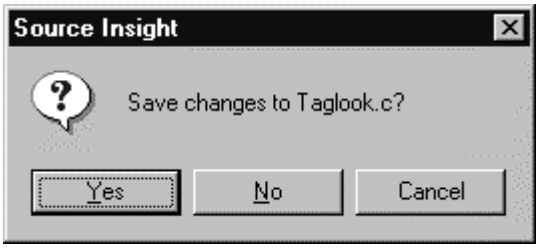
The Checkpoint File command is similar to the Save command. The save command preserves the undo and change history of a file. The Checkpoint File command does not. Checkpoint File is like a "clean" Save operation.

Save A Copy

Saves the current file to a new file, but does not replace or affect the current file. The newly saved file is left open as just another file buffer. This is a handy way to duplicate a file.

Save All

The Save All command saves all files that are open and have changed since they were saved last. For each file that has changed and requires saving, a dialog box is presented.



Yes

Click this button to save the file.

No

Click this button to not save the file.

Cancel

Click this button to stop the Save All command.

Save All Quietly

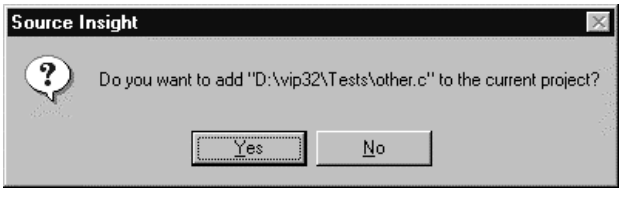
The Save All Quietly command saves all files that are open and have changed since they were saved last. Source Insight will not ask you if want to save each file; they will be saved automatically.

Save As

The Save As command saves the current file to disk as the name that you specify.

Adding a New File to the Current Project

If the file being saved is a new file that hasn't been saved before and you have a project open, then Source Insight will ask if you if you want to add the file to the current project.



Yes

Click to add the file to the current project.

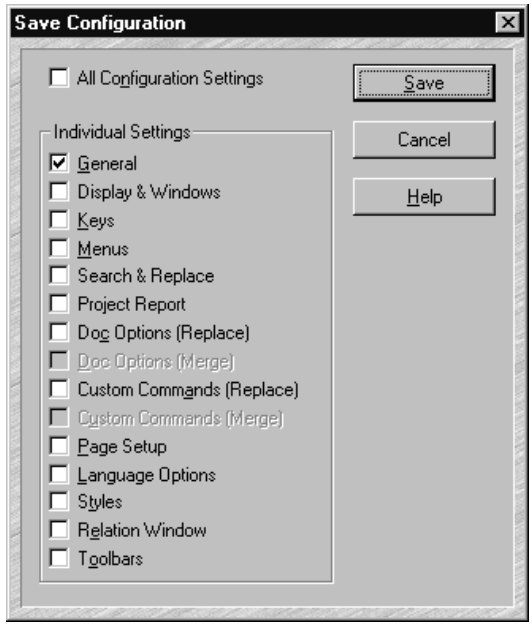
No

Click to not add the file to the current project. The file will still be saved.

Save Configuration

The Save Configuration command saves the current configuration to a configuration file that you specify. You can save the entire current configuration to a file, or just a specified subset of it.

When a configuration file is loaded that contains a configuration subset, it only affects the settings it contains. For example, you could save only keyboard settings to a configuration file and name it "MyKeyboard". When "MyKeyboard" is loaded, it will only affect the keyboard.



All Configuration Settings

If enabled, then all parts of the configuration are saved. Disable this to save only individual parts that are defined in the Individual Settings group below.

Individual Settings

Turning off the **All Configuration Settings** check box allows you to select the parts of the configuration you want to save. For example, this would allow you to save only the display settings,

such as screen colors and screen size, while leaving other parts of the configuration unspecified.

This grouping contains a check box for each configuration part. Check the items you want to save here.

Save

Displays the standard Save dialog box. You can select the configuration file you want to save to with this dialog box.

Having Multiple Configurations

You can keep several favorite configurations. After setting up the current configuration the way you like it in Source Insight, use the Save Configuration command to save each configuration to a different file. When you want to change configurations, use the Load Configuration command and specify the name of the configuration file you want to open. When the configuration file is opened, it replaces the current configuration.

Note: Once you load a configuration file, it will be automatically saved to the current configuration file that is in effect. By default, that file is *Global.cf3* in your Source Insight program directory. Make sure you make a backup of *Global.cf3* if you want to keep it!

See also: the Load Configuration command

Save Selection

The Save Selection command saves the currently selected text to a new file. The new file will remain open. The file may be a new file, or an already existing file. The file may also be a file that is already open in Source Insight. If the file is already open, then Source Insight will ask if you want to replace the file with the new text, or append the new text to the file.

Save Workspace

The Save Workspace command saves the current workspace to a workspace file that you specify.

The current workspace is automatically saved for you when you exit Source Insight, and reloaded the next time you run Source Insight.

Working With Multiple Workspaces

If you find that you work with sets of files, rather than individual files, you can save each set of files to a different workspace file.

When you want to change to another set of files, use the Open command and specify the name of the workspace file you want to open. When the workspace file is opened, it replaces the current workspace. In other words, all files are closed, and the files in the new workspace are opened.

Scroll Half Page Down

The Scroll Half Page Down command scrolls the active window down by half a window in distance.

Scroll Half Page Up

The Scroll Half Page Up command scrolls the active window up by half a window in distance.

Scroll Left

The Scroll Left command scrolls the active window to the left by one tab size.

Scroll Line Down

The Scroll Line Down command scrolls the current window down in the file by one line.

Scroll Line Up

The Scroll Line Up command scrolls the current window up in the file by one line.

Scroll Right

The Scroll Right command scrolls the active window to the right by one tab size.

SDK Help

The SDK Help command takes the word in the current selection and looks it up in the Windows Software Development Kit help file. For

example, if you selected "TextOut" in your program and ran the SDK Help command, a Help window for the TextOut Windows function would open.

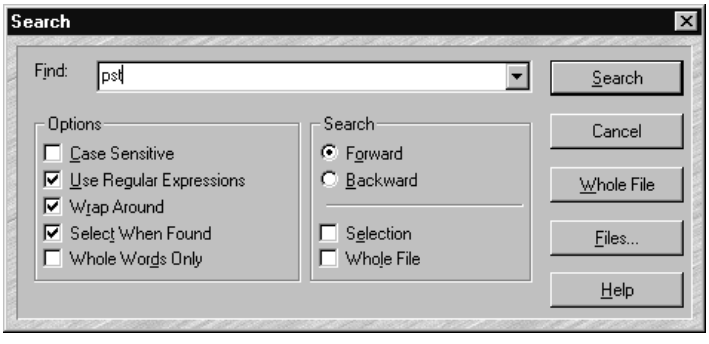
You must have a Windows SDK help file installed on your computer to use this.

Any help file for WinHelp 3.1 or greater may be used; it does not have to be an SDK help file. If you often want to perform help lookups from Source Insight using a different help file, that will work just fine.

You can tell Source Insight what WinHelp file to run for the SDK Help command by running the Change the SDK Help File command.

Search

The Search command searches the current file or selection for a specified pattern.



Find

Add the pattern you want to search for in this text box.

Search

Click this to begin searching.

Cancel

Click this to cancel the command.

Whole File

Click this button to search the whole file, from top to bottom, and place the search results in the Search Results window.

Files

Click this button to open the Search Files dialog box, which lets you search across files.

Case Sensitive

Search will only find matches if the case matches exactly.

Use Regular Expressions

The Find pattern is assumed a regular expression. See "Regular Expressions" on page 125.

Wrap Around

If checked, the search continues at the beginning of the file when it reaches the end of the file. The search will wrap around only once. If not checked, the search stops when it reaches the end of the file.

Select When Found

If checked, Source Insight will select any characters that match when a match is found. If not checked, Source Insight will put the insertion point before the first character of the matching text.

Whole Words Only

If checked, then Source Insight only finds matches that are whole words. If not checked, then Source Insight will also find matches that are embedded in words.

Search Scope

This group of options specifies the scope and the direction of the search.

Forward

Searches forward starting at the current selection. The search is always forward if **Selection** or **Whole File** is checked.

Backward

Searches backwards starting at the current selection.

Selection

Searches only the current selection, in the forward direction.

Whole File

Searches the whole file, in the forward direction.

If neither **Selection** nor **Whole File** is checked, then the search continues from the current selection point, either forward or backwards through the file.

Search Backward

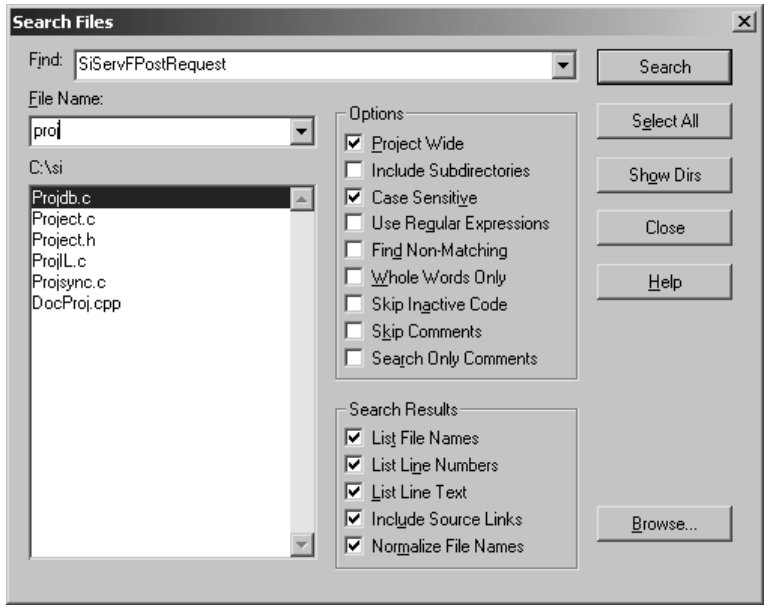
The Search Backward command searches backward in the current file for the pattern previously searched for. The search pattern is initially added using the Search command dialog box. The search begins at the current insertion point.

Search Backward for Selection

This command searches for the previous occurrence of the first word in the current selection. To use this command, put the insertion point within the word you want to search for and invoke this command. Source Insight will find the previous occurrence of that word.

Search Files

The Search Files command searches through multiple files. A new Search Results output window is created. Each time Source Insight finds a matching line in a file, it appends an entry to the Search Results. Each line in the Search Results file can have source links that link the line with the location of the matching text in another file.



Search

Click this button to begin the searching in the selected files, or the file named in the **File Name** text box.

Select All

Click to select all the files in the file list.

Browse

Click this button to show the Open File dialog box, so that you can browse your disks to locate a file to be searched. When you select a file in the Open File dialog box, its full path will be placed in the **File Name** text box.

Find

Type the pattern to be found in this text box. The pattern can be a regular expression.

File Name

The name of the file to search. You may also add a series of wildcard specifications and click the **Search** button and Source Insight will replace the file list with the results of the wildcard expansion. If the **Project Wide** option is on, the wildcards are expanded over the whole list of files in the current project; otherwise, the wildcards are expanded in the current directory.

If the **Project Wide** option is on, Source Insight will search the project symbol for file names added in the **File Name** text box, so you don't have to include a directory specification for those files.

File list

If the **Project Wide** option is on, then this list displays all files in the current project.

If the **Project Wide** option is off, then this list displays all the files in the current working directory. The current directory path is displayed above the file list. Source Insight shows only files for known document types in the current directory. The document types are specified with the Document Options command.

Show Dirs

Click this button to toggle the file list contents between showing file names, and showing only subdirectory names. When the subdirectories are shown, this button changes to "Show Files".

Options Group

Project Wide

This check box controls whether the File list shows all the files in the project, or just the files in the current working directory.

Include Subdirectories

If this check box is checked, then any selected directories are recursively searched. This option and the **Project Wide** option are mutually exclusive.

To recursively search a set of directories:

1. Uncheck the **Project Wide** check box.
2. Check the **Include Subdirectories** check box.
3. Select one or more directories in the file list.

You can also type a file wildcard specification in the **File Name** text box to limit the search to particular file extensions or names.

Case Sensitive

If checked, Source Insight will only find matches if the case matches exactly.

Use Regular Expressions

If checked, the Find pattern is assumed a regular expression. . See "Regular Expressions" on page 125.

Find Non-Matching

If checked, Source Insight will find all lines where the pattern **did not** match.

Whole Words Only

If checked, then Source Insight only finds matches that are whole words. If not checked, then Source Insight will also find matches that are embedded in words.

Skip Inactive Code

If enabled, then only code that is active under conditional compilation is searched. You must first specify known conditions in the Preferences: Languages dialog box, in order for Source Insight to know what conditions are active or not. Conditional compilation only applies to some languages.

Skip Comments

If enabled, then comments will not be searched.

Search Only Comments

If enabled, then *only* comments will be searched. This is mutually exclusive with the **Skip Comments** option. The comment options slow the search down a little.

Search Results

These options affect what appears in the Search Results after the search is completed.

List File Names

If checked, then the file name where the match was found is inserted in the search results. Note: If this option is on, and **List Line Numbers** and **List Line Text** are both off, then a file name is inserted in the search results only once if any matches were found in the file. That is, only one match per file is listed in the search results.

List Line Numbers

The line number in the file where the match was found is listed in the search results.

List Line Text

The source text of the line where the match was found is listed in the search results.

Include Source Links

If checked, then source links are also created for each line appended to the search results. Source links allow you to jump between the line in the search results and the line in the file where the match was found. Source Links are adjusted automatically while you edit, so they maintain their linkage.

If not checked, then only text is appended to the search results. You may want to turn off this option if you think you will find thousands of matches, since source links take up memory.

Normalize File Names

If checked, then the file names listed in the search results will be normalized. If not checked, then the file names listed will be full paths. See "Normalized File Names" on page 81.

To Search a Set of Files

If you add file name wildcards into the **File Name** text box and click the **Search** button, the wildcard list will be expanded, and all the files in the file list will be selected automatically. If the **Project Wide** check box is on, then the wildcards will be expanded over all files in the project.

So, for example, if you wanted to search all .h files in your project, you would add `*.h` into the **File Name** text box, press Enter to click the **Search** button, and click the **Search** button again to search all the files in the file list. If the **Project Wide** option is on, Source Insight will fill the file list with all .h files in the project, regardless of what directory they are in.

See Also: "Replace Files" on page 317.

Search Forward

The Search Forward command searches forward in the current file for the pattern previously searched for. The search pattern is initially added using the Search command or the Search Forward for Selection command. The search begins at the current insertion point.

Search Forward for Selection

This command searches for the next occurrence of the first word in the current selection. To use this command, put the insertion point within the word you want to search for and invoke this command. Source Insight will find the next occurrence of that word.

Search List

The Search List command appears on most right-click menus when you click on a list. This allows you to search the list for a string or regular expression.

Note: Once the input focus is on a list, you can press F4 to search for the next occurrence.

String to find in the list

Add the string pattern to search for.

Start at beginning

If checked, then the search starts at the first item in the list. If not checked, then the search starts just after the selected item in the list.

Match Case

Turn this on to perform a case-sensitive search.

Use Regular Expressions

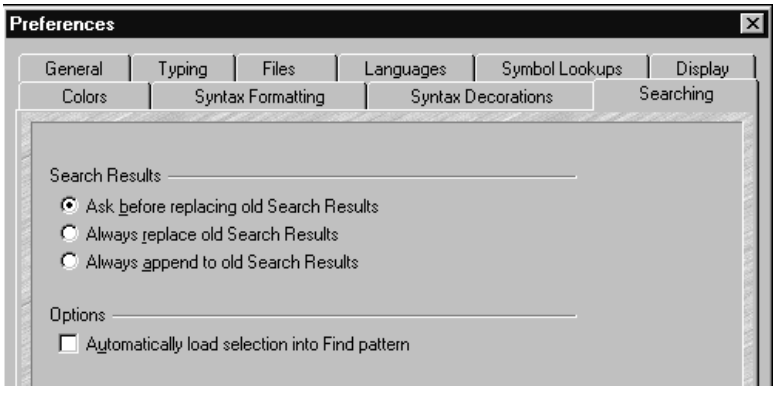
Turn this on to interpret the search string as a regular expression. See "Regular Expressions" on page 125.

Search Project

Searches for text or keywords across all project files. This command works the same as the Lookup References command. The only difference is that each dialog box has its own persistent state. See "Lookup References" on page 272.

Searching Options

Specifies options for the Search commands.



Search Results

These options control how text is added to the Search Results window.

Ask before replacing old Search Results

You are always prompted as to whether you want to replace or append to the existing Search Results, or to create a new Search Results file.

Always replace old Search Results

Any existing Search Results are thrown out and replaced with the new results.

Always append to old Search Results

The new search results are appended to the end of the current Search Results file as a new results set.

Automatically load selection into Find pattern. The word under the cursor is automatically loaded into the Find pattern of the Search dialog boxes. If disabled, then the previous search pattern is preserved.

Select All

The Select All command selects all the text in the current file.

Select Block

The Select Block command selects the smallest C block that encloses the current selection. Each time the Select Block command is used, it selects the next larger C block.

Select Char Left

The Select Char Left command extends the current selection left by one character.

Select Char Right

The Select Char Right command extends the current selection right by one character.

Select Function or Symbol

The Select Function or Symbol command attempts to select the whole enclosing symbol, such as the enclosing function. You can also invoke this command by double clicking the mouse in the left margin.

Select Line

The Select Line command selects all of the current line.

Select Line Down

The Select Line Down command extends the current selection down by one line.

Select Line Up

The Select Line Up command extends the current selection up by one line.

Select Match

The Select Match command selects up to the matching brace, parentheses, or quote mark. For example, if the insertion point is just before an open brace, this command selects up to and including the closing brace.

Select Next Window

The Select Next Window command changes the active window focus to the next window. This command cycles through all open windows.

If the active window is maximized when this command is used, the next window is shown as maximized also.

Select Paragraph

The Select Paragraph command selects the entire enclosing paragraph. A paragraph of text is assumed to be a series of lines, bounded by blank lines.

Select Sentence

The Select Sentence command selects up to the next period.

Select Symbol

The Select Symbol command selects the entire enclosing symbol. For example, if the current selection is inside of a function, the Select Symbol command selects the whole function, including the lines that precede the function, up to the bottom of the previous symbol.

Select To

The Select To command is used with the mouse to extend an existing selection up to a new point.

To use this command, point and click the left mouse button while holding down the Shift key. The selection will be extended up to the place you pointed at. If you pointed to a position already within the selection, then the selection will be shrunk to that location.

Select To End Of File

The Select To End Of File command extends the selection from the insertion point to the end of the file.

Select To Top Of File

The Select To Top Of File command extends the selection from the insertion point to the beginning of the file.

Select Word

The Select Word command selects the whole word at the insertion point.

To use this command with the mouse:

1. Point and click the left mouse button at a word while holding the Ctrl+key down.
2. The whole word gets selected. Now, while still holding the left button down, you can drag and extend the selection in whole word increments.

Select Word Left

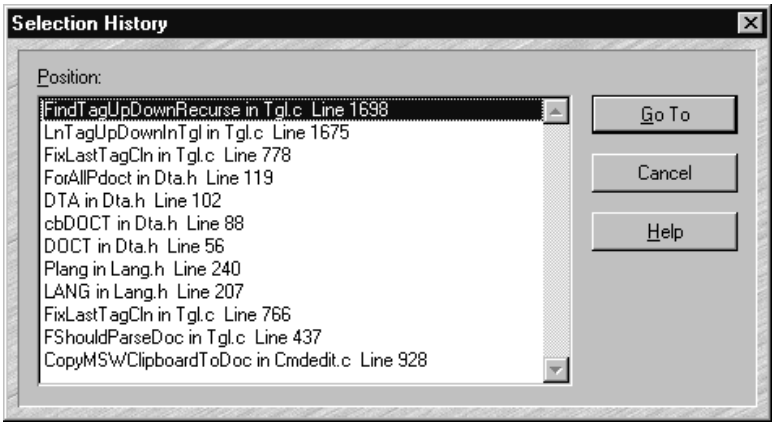
The Select Word Left command extends the selection from the insertion point to the beginning of the current word.

Select Word Right

The Select Word Right command extends the selection from the insertion point to the end of the current word.

Selection History

The Selection History displays a list of places that you have been in the currently open files. The selection history is part of the current workspace.



Position

Displays a list of all selection history positions. Each item in the list shows the file and line number. If the position is within a symbol, the symbol is also shown. For example, if you were inside of a function, then the function name is in the list too.

Go To

Click this button to jump to the selected position.

Setup Common Projects

This command asks you to indicate what common projects you would like to build.

This command runs automatically after you install Source Insight for the first time. You can also invoke this command directly at any time after that from the Preferences: Symbol Lookups dialog box.

What Are Common Projects?

Most Source Insight users make use of standard libraries, such as the C/C++ runtime libraries, or the standard Java packages. In order for Source Insight to provide symbol completion, and other symbolic features for standard libraries, you need to setup separate projects for those libraries. Source Insight will resort to searching these common projects if a symbol cannot be found in your current project.

The Setup Common Projects command will help you build projects for those libraries. The projects you build are added to the project symbol path, so Source Insight can provide symbol completion and

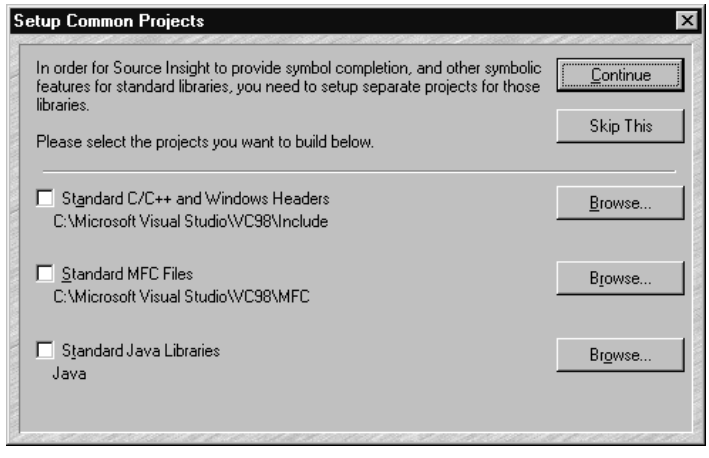
other symbolic features for those libraries from within your own projects.

Set Common Projects Dialog box

For each common project, you are asked to locate the directory where the corresponding files are located on your disk. If you installed the source code for your libraries on your disk, then you can take advantage of Source Insight to use the source code as a basis for the projects. For instance, you might click on the C function `strtok`, and Source Insight will locate the source code for `strtok`.

Each common project created here is appended to the project symbol path, which can also be edited in the Preferences: Symbol Lookups dialog box.

Note: Selecting a project to rebuild in this command will replace the existing project, if any. Also, building a large project, such as the C/C++ Runtime and Windows project may take a few minutes.


Setup Common Projects
In order for Source Insight to provide symbol completion, and other symbolic features for standard libraries, you need to setup separate projects for those libraries.
Please select the projects you want to build below.
Continue
Skip This
Standard C/C++ and Windows Headers
C:\Microsoft Visual Studio\VC98\Include
Browse...
Standard MFC Files
C:\Microsoft Visual Studio\VC98\MFC
Browse...
Standard Java Libraries
Java
Browse...

Standard C/C++ and Windows Headers

This project is intended to include the standard Windows, C, and C++ include files, and/or source code. Source Insight will attempt to locate them by looking in your registry, however you will have to confirm their location.

Standard MFC Files

This project is intended to include the MFC (Microsoft Foundation Classes) include files and/or source files.

Standard Java Libraries

This project is intended to include the Java development kit source code for the standard Java packages, such as java.lang. Source Insight will attempt to locate them by looking in your registry, however you will have to confirm their location.

Continue

Click the Continue button to proceed and create the common projects that are checked. For each project that is checked, Source Insight will ask you to add files to that project. The Add and Remove Project Files dialog box will appear.

Setup HTML Help

Use this command to locate the HTML Help file on your disk that will be used by the HTML Help command. If you have Microsoft® MSDN™ or Microsoft® Developer Studio™ tools installed, you will probably want to select the compiled HTML help "collection" file that is the main help file for the developer tools. The file has a .col extension. That will allow you to invoke HTML Help on Windows development APIs from within Source Insight.

Setup WinHelp File

This command allows you to locate the WinHelp help file on your disk to be used for the SDK Help command. A system Open File dialog box will appear and allow you to pick the .HLP file to be used.

Show Clipboard

The Show Clipboard command opens a window, which displays the clipboard. You cannot edit or select in the clipboard.

Show File Status

The Show File Status command shows the current file's size in lines and bytes in the status bar. It also shows whether the file has been changed since it was saved last, and if it is read-only.

Simple Tab

Simple Tab Inserts a regular tab, overriding the Smart Tab mode. This is useful if you have the Smart Tab option enabled. The Smart Tab mode alters the behavior of the regular Tab key. Sometimes, the Smart Tab results in unwanted results. Use the Simple Tab command to just insert a regular tab, without any special effects.

Smart End of Line

Moves the cursor generally to the end of the line. It does one of the following:

1. If the cursor is in the middle of a line, move it to just after the last non-white space character on the line.
2. If the cursor is after the last non-white space character, move it to the actual end of the line.
3. If the cursor is already at the end of the line, move to the end of the file.

Smart Beginning of Line

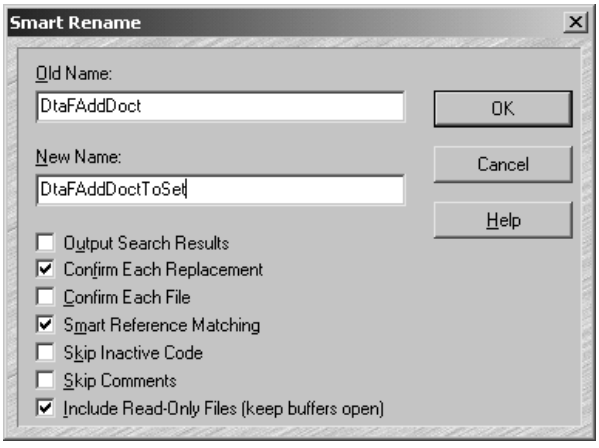
Moves the cursor generally to the beginning of the current line. It does one of the following:

1. If the cursor is in the middle of a line, move it to just before the first non-white space character on the line.
2. If the cursor is before the first non- white space character on the line, move it to the actual start of the line .
3. If the cursor is already at the start of the line, move to the start of the file.

Smart Rename

Smart Rename will rename a symbol. If the Smart Reference Matching option is enabled, then Smart Rename will rename the symbol only in the correct scope contexts. It can rename a symbol across all project files. It can be used to rename function local variables, class or struct member, and functions.

Smart Rename is a specialized form of a global search & replace. Source Insight uses its symbol database index to make it very fast.



Old Name

Add the name of the identifier to be renamed. The word under the cursor is automatically loaded for you. The position of the cursor is significant because Source Insight will determine exactly which symbol you want to rename, based on the local scope context.

You can add any string into this text box; however, the rename operation is optimized and *much* faster for single-word strings. Also, if you type anything into this text box, Source Insight will have to re-establish exactly what symbol you are trying to rename, based on the initial cursor position.

If you are renaming a member variable, or a local variable, you will notice that the Old Name text box contains the full symbol name, including the container symbols. For example, it might say "DocDraw.paintStruc", where "DocDraw" is a function name, and "paintStruc" is a local variable. In a sense, "paintStruc" is a member of the "DocDraw" function.

New Name

Add the new name here. For members, you should only add the new member name, and omit the symbol container qualifiers.

Output Search Results

If checked, then the results of the search will be output to the Search Results window. This provides you with a log of changes made to each occurrence. The Search Results window will list the text the way it was *before* the replacement with the **New Name** string.

Confirm Each Replacement

If checked, Source Insight will confirm each replacement by prompting you.

Confirm Each File

If checked, Source Insight will confirm each modified file by prompting you.

Smart Reference Matching

This tells Source Insight to use its language information, and the cursor scope context to determine exactly what symbol is being renamed, and to make sure it only renames strict references to it.

Skip Inactive Code

If enabled, then only code that is active under conditional compilation is searched. You must first specify known conditions in the Preferences: Languages dialog box, in order for Source Insight to know what conditions are active or not. Conditional compilation only applies to some languages.

Skip Comments

If enabled, then symbol references inside comments will not be renamed.

Include Read-Only Files (keep buffers open)

If checked, then replacements will be made inside of read-only file buffers. Source Insight will not attempt to save the file as the replacement operation progresses. The files will be left open and modified, allowing you to save the files yourself. If not checked, then read-only files will be skipped. Note that this options works independently from the Preferences: Files option **Allow editing read-only file buffers**.

You can make Source Insight automatically save over read-only files while renaming if you enable the Preferences: Files option: **Save over read-only files without prompting**.

Smart Tab

When the Smart Tab command is used at various positions in your source code, Source Insight moves the selection to the next "field". A field is an interesting position, depending on the current context. Smart Tab lets you move the cursor around easily, especially when typing new function calls.

When the Smart Tab option is on (Preferences: Typing), then pressing the regular Tab key will invoke the Smart Tab command. The Simple Tab command simply inserts a tab, and avoids the Smart Tab behavior. Therefore, if you have the Smart Tab option

turned on, you can use the Simple Tab command to occasionally override the Smart Tab behavior.

Smart Tab Examples

Here are some examples, starting with step 1 as the initial selection state, and steps 2 and later are the new selections after using Smart Tab. The current insertion point is marked by ^ and a selected range of text is bracketed `[[like this]]`:

E.g. 1:

```
1. Beg^inPaint(hwnd, pps);
2. BeginPaint([[hwnd]], pps);
```

E.g. 2:

```
1.      BeginPaint^(hwnd, pps);
2.      BeginPaint([[hwnd]], pps);
3.      BeginPaint(hwnd, [[pps]]);
4.      BeginPaint(hwnd, pps^);
5.      BeginPaint(hwnd, pps)^;
```

E.g. 3:

```
1.      ResetAbc(^)
2.      ResetAbc()^
```

Smart Tab works like a regular tab when you use it at the beginning or end of line text, or on a line that doesn't have a function call.

The Smart Tab works well with auto-completion of function calls. When you insert a function call via the popup auto-completion window, the function's parameter types and names are also inserted, and the first parameter is selected. You only have to start typing over the first parameter, then press Smart Tab to select the next parameter.

Sort Symbol Window

The Sort Symbol Window command cycles the sorting state of the symbol window in the current file window. The sort states are:

1. by Name.
2. by Line number (the default).
3. by Type + Name.

Sort Symbols By Line

Sorts the symbol entries listed in the Symbol Window by line number. Each symbol in the file will appear in the list in the order of occurrence.

By default, the Symbol Window is sorted by line number (occurrence).

If you want all Symbol Windows to be sorted this way by default, then right-click on the Symbol Window and run the Record New Default Properties command on the Symbol Window's right-click shortcut menu.

Sort Symbols by Name

Sorts the symbol entries listed in the Symbol Window alphabetically by symbol name.

By default, the Symbol Window is sorted by line number (occurrence).

If you want all Symbol Windows to be sorted this way by default, then right-click on the Symbol Window and run the Record New Default Properties command on the Symbol Window's right-click shortcut menu.

Sort Symbols By Type

Sorts the symbol entries listed in the Symbol Window by symbol type. For example, all structs will appear together, followed by all functions, etc.

By default, the Symbol Window is sorted by line number (occurrence).

If you want all Symbol Windows to be sorted this way by default, then right-click on the Symbol Window and run the Record New Default Properties command on the Symbol Window's right-click shortcut menu.

Source Dynamics on the Web

This command opens your web browser and goes to the Source Dynamics web site.

Start Recording

The Start Recording command turns on the command recorder. While recording, any command you run will also be recorded. This allows you to record a single series of commands, which can be played back with the Play Recording command.

To stop recording, use the Stop Recording command, or just play the recording back with the Play Recording command.

You can keep only one recording at a time. The recording is saved with the workspace.

Stop Recording

The Stop Recording commands turns off the command recorder. The Start Recording command is used to start the recorder. The command recorder allows you to record a single series of commands, which can be played back with the Play Recording command.

Style Properties

This command allows you to set formatting properties for display styles. For more information about how styles work, see

"Syntax Formatting and Styles" on page 111.

Formatting Properties

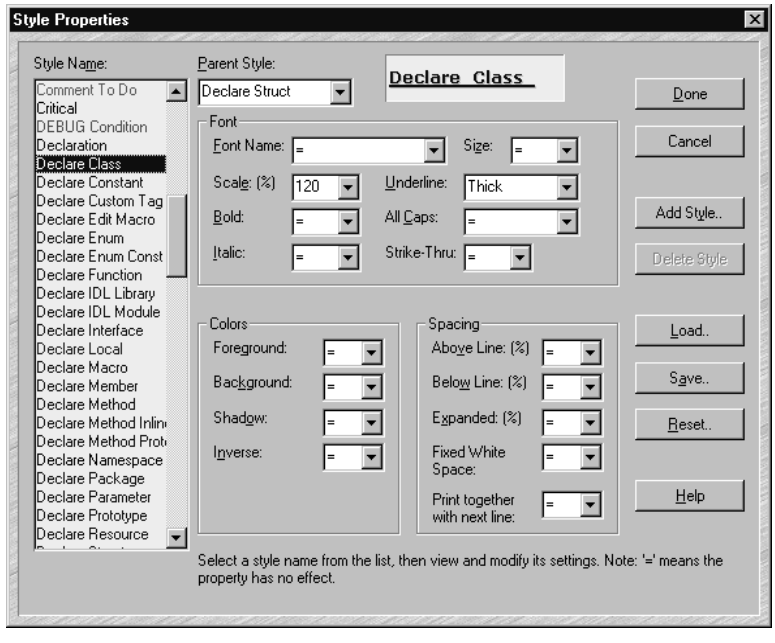
Style properties are combined with the parent style.

Each style has a number of formatting properties. Because styles exist in a hierarchy, each formatting property is combined with the parent style to yield a final result.

For example, if bold = "ON", then bold formatting is added. If bold = "OFF", then bold formatted is subtracted from the parent style properties.

Many formatting controls in this dialog box show one of these values:

- **On** – the property is added to the parent style formatting.
- **Off** – the property is deleted from the parent style formatting.
- **A Number** – the value replaces the parent style property.
- **= (equal)** - the property has no effect, and it inherits the exact same value as in the parent style.



Style Name list

Lists all the syntax formatting styles. When you select a style in this list, its properties are loaded into the controls to the right. A sample of the style is also displayed in the sample box.

Parent Style

This is the parent style in the style hierarchy. The current style inherits its formatting from the parent style.

Add Style

Click this button to add a new user-defined style.

Delete Style

Click this button to delete a user-defined style. The standard built-in styles cannot be deleted.

Load...

Click this button to load a new style sheet from a configuration file.

Save

Click this button to save the current style sheet settings to a new configuration file. The file will contain only style properties, and won't contain other elements that can be stored in a configuration file. If you load this configuration file, only the style properties are altered.

Reset...

Click this button to reset all the styles to the factory defaults. This loses all your changes since installing Source Insight.

Font Options

Font Name

Indicates the font currently selected.

Size

Selects the font size, specifically as a point size. You may find the relative **Scale** property more useful, since it is relative, and works well regardless of changes to the parent styles.

Scale

Specifies the font size scaling as a percentage of the parent style's font size. For example, if the scale is 50%, then it will be half the size of whatever the parent style font size is.

Bold

Selects the bold property of the style, if any.

Italic

Selects the italic property of the style, if any.

Underline

Selects the underline property of the style, if any.

All Caps

Selects the All Caps (capitalization) property of the style.

Strike-Thru

Selects the Strike-Thru property of the current style.

Colors Options

Foreground

Selects the foreground color of the current style.

Background

Selects the background color of the current style.

Shadow

Selects the color of the drop-shadow of the current style.

Inverse

Selects the Inverse property of the current style. Inverse means that the foreground and background colors are reversed.

Spacing Options

Above Line

This selects the percentage of vertical spacing to add above the line.

Below Line

This selects the percentage of vertical spacing to add below the line.

Expanded

This selects the percentage of horizontal spacing to add to characters.

Fixed White Space

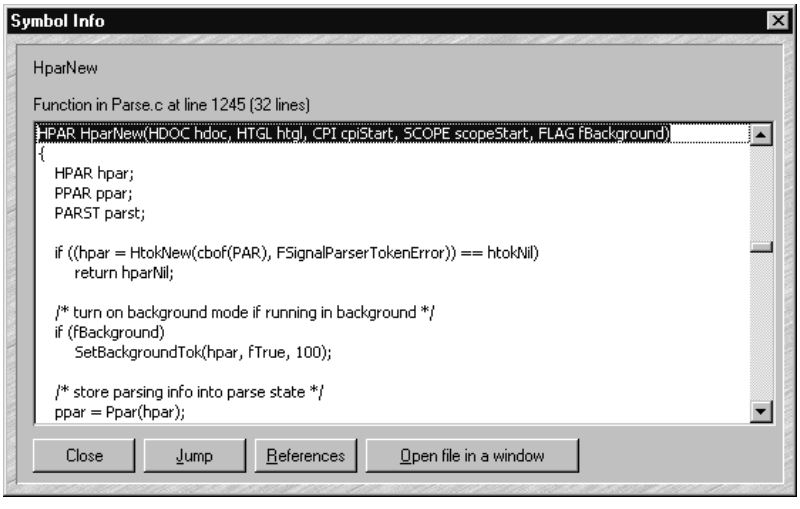
This option only applies if you have selected a proportionally spaced font. Fixed-pitch fonts, such as Courier New, are not affected. If enabled, Source Insight will attempt to use a fixed width for spaces and tabs so that tabs line up the same way they do with a fixed-pitch font. Programs generally look better with this turned on if you are using a proportional font. See also: "Character Spacing Options" on page 215.

Print together with next line

If enabled, Source Insight will try to keep the text on the same page as the following line, when printing.

Symbol Info

The Symbol Info command displays a pop-up window showing the definition of the symbol under the cursor. This is a quick way to check the definition of an identifier.



Symbol Name, Type, and Location

The symbol's name, type, and location are displayed at the top of the window.

Source File

The source file name and line number where the symbol is defined are displayed at the top left of the dialog box below the symbol name. If known, the symbol's size in lines is displayed also.

Text Window

This scrollable window contains the contents of the source file where the symbol is defined.

Close

Click this to close the window.

Jump

Click this to close the Symbol Info window and jump to the symbol definition directly.

References

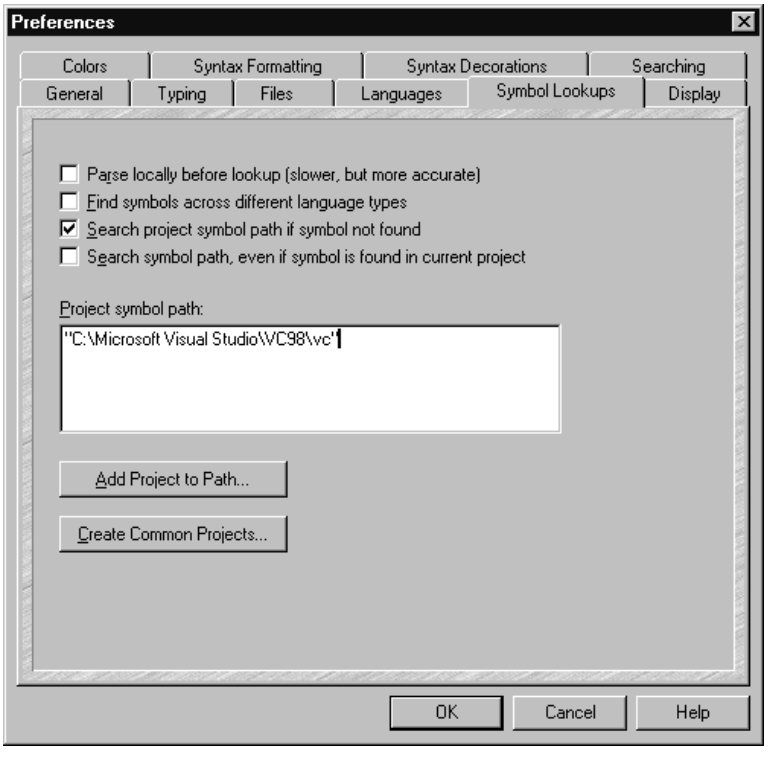
Click this button to search for references in the whole project to the symbol.

Leave File Open

Click this button to leave the file displayed in the list box open. This button is disabled if the file is already open. If you leave the file open, you can select the file name from Window menu after the window closes.

Symbol Lookup Options

Sets options for the way Source Insight looks up symbol definitions. This command activates the Symbol Lookups page of the **Preferences** dialog box.



Parse locally before lookup

Source Insight will ensure that the symbolic information for the current file is completely up-to-date before trying to lookup a symbol. This option comes into play when you are editing. Every time you type a character, Source Insight considers the symbol data for the file to be stale. If this option is enabled, then the file will be parsed after you type anything. If the option is disabled, then Source Insight will use the possibly stale symbol data for the current file. Enabling this option will make the symbol lookups more accurate, but it is slower. It will also cause the auto-completion window to appear slower. Most of the time, lookups work fine when this option is turned off.

Find symbols across different language types

If checked, then Source Insight will lookup symbol definitions in any language, regardless of the source language. If unchecked (the default) then only symbols that are defined in the same language will be found.

Search symbol path if symbol is not found

If a symbol cannot be found in the current project, or any open file, then Source Insight will search the projects listed in the project

symbol path. If it does search the symbol path, then it searches through *all* projects in the symbol path.

Search symbol path, even if symbol is found in current project

When enabled, all projects in the symbol path are searched *every time* Source Insight looks up a symbol, even if the symbol was already found in either an open file or the current project. This is sometimes useful if you are working on an alternate version of a project, with many of the same symbol names. Looking up a particular symbol in the current project will also show matches in the other projects on the symbol path.

If disabled, then Source Insight will only search the symbol path if a symbol was not already found in either an open file or the current project.

Project symbol path

The project symbol path is a delimited list of projects that Source Insight will search through when looking up a symbol. The project symbol path enables you to create smaller, self-contained projects, but still have the ability to locate symbols in other projects.

Each item in this list should be a full path name of a project. Project paths should be separated by a semi-colon.

Example:

```
c:\include\include;c:\windev\include\include
```

The project symbol path is only used for locating the definitions of symbols external to the current project. It is not used for finding references to symbols, or for searching across multiple projects.

Add to Project Path...

Click this button to pick a project to append to the exiting project symbol path.

Create Common Projects...

Click this button to open the Create Common Projects dialog box, where you can create the common external projects that will be helpful to have on the project symbol path.

Symbol Window command

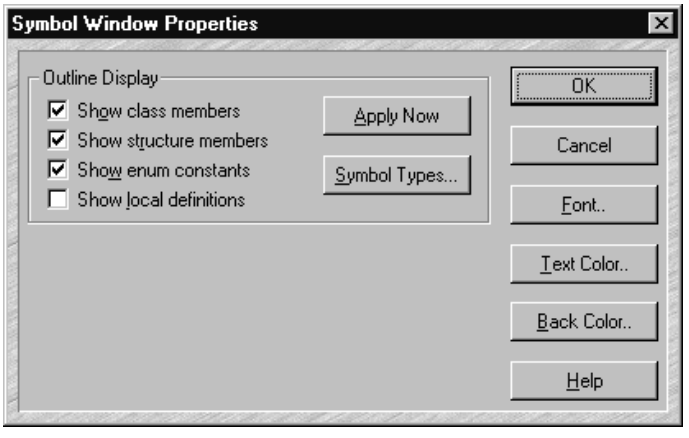
The Symbol Window command opens and closes the symbol window in the current file window. The symbol window is a vertical list of symbols defined in the file that are normally visible at

global scope. The symbol window can be sorted by using the Sort Symbol Window command.

The symbol window is only available if the file's document type has a parsed language selected.

Symbol Window Properties

Displays the properties of the Symbol Window, which appears on the left of each source window.



Show class members

If checked, the member contents of classes are included in the Symbol Window. If unchecked, then only the class name is included in the list.

Show structure members

If checked, then the structure field members are included in the Symbol Window. If not checked, then only the structure name is included in the list.

Show enum constants

Includes enum constants in the Symbol Window. If not checked, then only the enum type name will appear in the list.

Show local definitions

If checked, then function local declarations are also shown in the Symbol Window under each function.

Apply Now

Applies the changes you made in this dialog box to the Symbol Window.

Symbol Types...

Click this button to open the Symbol Type dialog box, where you can filter out different types of symbols from the Symbol Window.

Font...

Click this button to pick the font used to draw the Symbol Window.

Text Color...

Click this button to pick the color of the text in the Symbol Window.

Back Color...

Click this button to pick the background color of the Symbol Window.

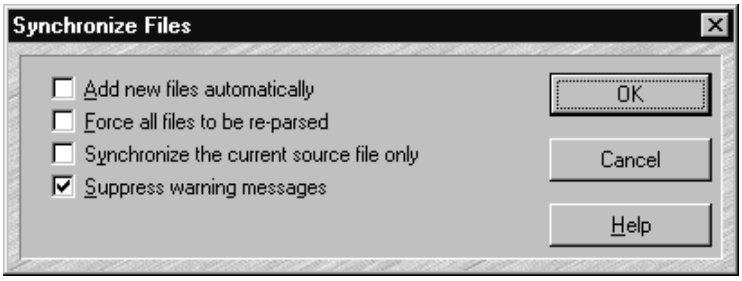
Sync File Windows

The Sync File Windows command scrolls all windows showing the current file to same location as the current window.

Synchronize Files

The Synchronize Files command synchronizes the current project with all the source files in the project. The command scans each file in the project and updates the symbol database for each file that has been modified since Source Insight parsed the file last. In addition, files that were part of the project that don't exist anymore are removed from the project.

As an alternative, you can have the synchronization happen in the background, while you edit, by turning on the **Background project synchronization** option in the Preferences: General dialog box.



Add new files automatically

Before synchronizing all the files, Source Insight will add new files in the project's root directory and in all subdirectories, recursively. However, only directories that already have project files in them are scanned. Directories that are not descendents of the project root directory are not scanned. This feature allows you to add new files to your project directories, and then run the Synchronize Files to add those new files to your Source Insight project automatically.

Force all files to be re-parsed

If checked, then Source Insight will ignore file time-stamps and consider *all* files in the project to be out of date. It will update the symbol database for all files. This provides an easy and relatively quick way to completely rebuild Source Insight's project data files.

If not checked, then only those files in the project that are considered out of date are updated.

Synchronize the current source file only

Only the currently active source file is synchronized with the symbol database. This has the effect of replacing all symbol database information known for the current source file.

Suppress warning messages

Source Insight will not inform you of problems, such as not being able to open or read a file.

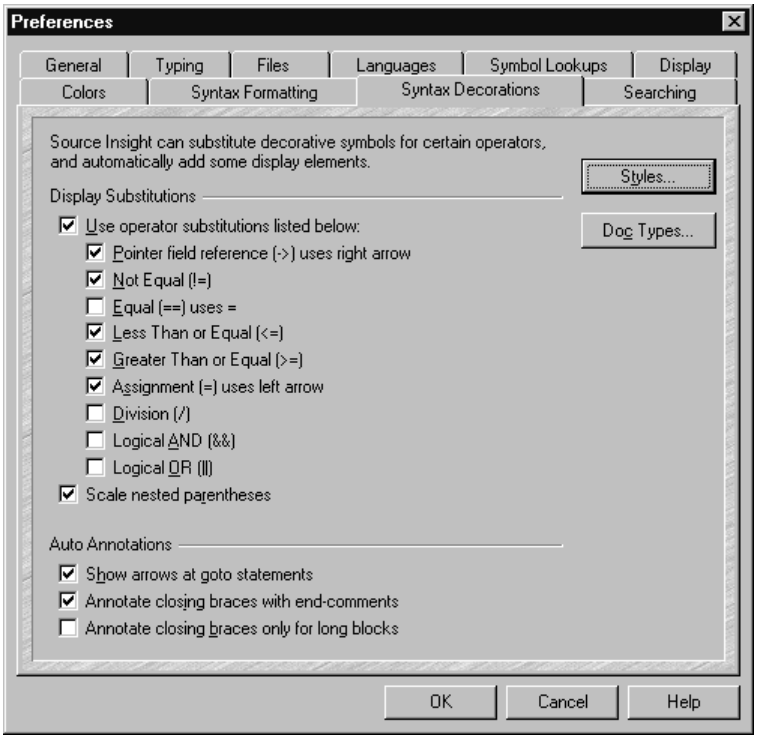
Syntax Decorations

This command specifies syntax decoration options for displaying source files. It activates the Syntax Decorations page of the **Preferences** dialog box.

Source Insight can replace some common operators with more useful symbolic characters. The Syntax Decorations command lets you control which decorations are used.

It's important to remember that symbol decorations and substitutions do not change the text in the source file; only its representation on the screen changes to show the special symbols. You still need to type the operators normally when editing your code, or when searching for them.

For more information, see "Syntax Decorations" on page 116.



Display Substitutions

This group of options controls which operators are substituted with special symbols.

User special operator symbol substitutions

This enables or disables the whole group of operator substitutions that appear below.

Pointer field reference uses right arrow

This replaces the `->` pointer operator `with an actual arrow like this` →.

Not Equal

This replaces the `!=` not equal operator with ≠.

Equal

This replaces the == equal operator with =.

Less Than or Equal

This replaces the `<=` operator with ≤.

Greater Than or Equal

This replaces the `>=` operator with ≥.

Assignment uses left arrow

This replaces the = assignment operator with ←.

Division

This replaces the `/` division operator `with` ÷.

Logical AND

This replaces the `&&` logical And with ∩.

Logical OR

This replaces the `||` logical Or with ∪.

Scale nested parentheses

Nested parentheses are shown so that outer levels are larger than inner levels. This makes it easier to visually identify matching parentheses.

```
i ← (ITIR) (((UINT)iMin + (UINT)iLim) >> 1);
```

Auto Annotations

Source Insight can automatically add certain annotations to your source code display. The following options control which annotations appear.

Show arrows at goto statements

This causes either an up or down arrow symbol to appear in goto statements next to the label name. The arrow indicates whether the indicated label is above or below the goto statement line.

```
krel ← krelCalls;
goto ↑LSet;
```

Annotate closing braces with end-comments

This causes end-comment annotations to appear after closing curly braces. The end-comment contains a show description of the start of the block. For example:

```
            } « end switch *pch »
        } « end if stcNewWord!=stcString... »
    } « end if !FWhiteCh(*pch) »  // !fWhiteCh
```

Annotate closing braces only for long blocks

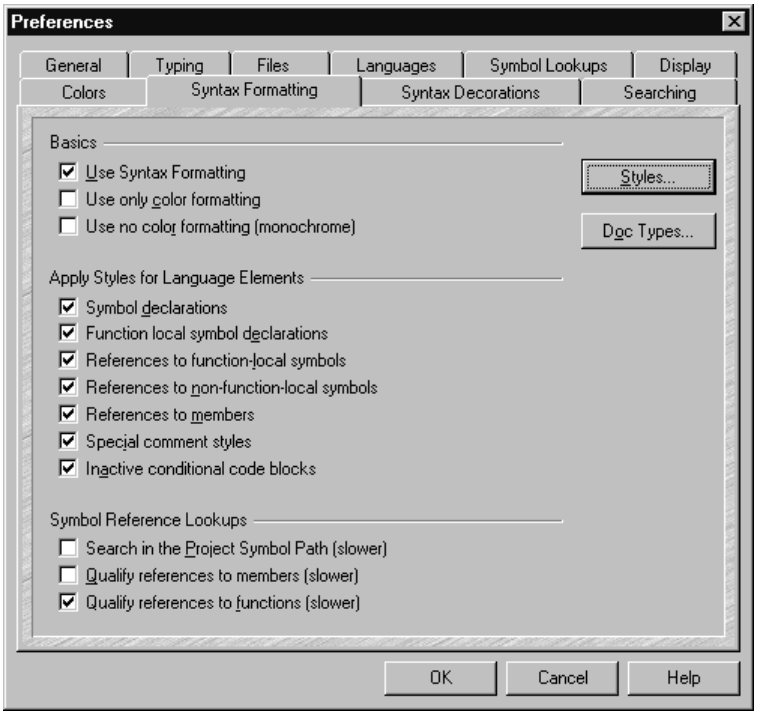
This suppresses the auto-annotated end-comments for blocks less than 20 lines tall.

Syntax Formatting

This command specifies syntax formatting options for displaying source files. It activates the Syntax Formatting page of the **Preferences** dialog box.

Source Insight uses information gathered from its language parsers to format source code. Identifiers can be displayed in different fonts or font sizes, along with a variety of effects such as bold and italics.

Formatting is applied using "styles". A style is a set of formatting properties. For example, a style may specify bold + italic. You can edit each style's formatting properties with the Style Properties command.



Styles...

Click to edit the style properties.

Doc Types...

Click to edit the document types.

Basics

Use Syntax Formatting

Enable this to have Source Insight display source code with syntax formatting. If disabled, then source code will display with no coloring or font changes.

Use only color formatting

All non-color formatting properties, such as font size changes, or bold and italics, will be suppressed. Display tokens will only contain color changes. This is similar to how earlier versions of Source Insight displayed text.

Use no color formatting (monochrome)

If enabled, then Source Insight will suppress all color changes. Text will be displayed in black and white and gray.

Apply Styles for Language Elements

Source Insight will apply styles to display tokens based on their lexical and contextual meaning. Each option in this group enables successively more elaborate formatting.

Symbol declarations

Declarations of symbols are formatted with the appropriate "Declare…" styles. For example, a function name will appear in the "Declare Function" style where it is declared.

Function-local symbol declarations

Declarations of local function scope variables and other symbols will be formatted with the appropriate "Declare…" styles. This includes local variables, and function parameters.

References to function-local symbols

References to local function scope variables and symbols are formatted with the appropriate "Ref to…" reference styles. For example, references to (i.e. usages of) a local variable will have the "Ref to Local" style.

References to non-function-local symbols

References to symbols declared outside of function scopes, such as class scopes and the global scope, are formatted with the appropriate "Ref to…" reference styles. This option requires more work, and it will slow down the display somewhat. The reference information is cached, so once a piece of code is rendered, it usually will display quickly afterwards.

References to members

References to structure and class members are formatted with the "Ref to Member" style. The veracity of the member reference can be controlled with the **Qualify references to members** option.

Special comment styles

Source Insight supports special comment styles that are controlled by special `//1-4` comment heading tokens, and the placement of comments. If this option is enabled, then Source Insight will apply the appropriate comment style to those special comments

Comment Headings

Comment heading styles are comments that begin with a single digit in the range 1 to 4. For example:

```
//1 This is heading one.
//2 This is heading two.
```

When the comment styles are used, the //x at the beginning of the comment is hidden, and the heading style formatting is applied to the rest of the line.

Inactive conditional code blocks

Code contained in inactive conditional code blocks are formatted with the "Inactive Code" style. An inactive code block is one contained in an inactive #ifdef, #if, #elif, or #else branch. You control the state of the conditions with the Edit Condition command.

Symbol Reference Lookups

When a potential reference is encountered, Source Insight must verify that the symbol is declared somewhere. This section controls how Source Insight resolves references to symbols declared in the project, as it renders source code.

Search in the Project Symbol Path

Source Insight will search not only the current project for a declaration, but also all the projects in the Project Symbol Path.

Qualify references to members

If enabled, Source Insight will verify that the member declaration exists before formatting it with the "Ref to Member" styles.

If disabled, then Source Insight will format tokens with the "Ref to Member" style if the tokens look syntactically like a member reference. There is no guarantee that the member actually exists. For example:

```
PtrFoo->somemember  // somemember looks like a member
reference
FooThing.somemember // somemember looks like a member
reference
```

Qualify references to functions

If enabled, Source Insight will verify that the function declaration exists before formatting it with the "Ref to Function" or "Ref to Method" styles.

If disabled, then Source Insight will format tokens with the reference styles if the tokens look syntactically like a function call. There is no guarantee that the function actually exists. For example:

```
SomeFunction(x) // SomeFunction looks like a function
reference
```

Tile Horizontal

The Tile Horizontal command arranges all windows so they are not overlapping. The tiling algorithm will attempt to make windows wider than they are tall. This is useful for viewing 2 or more files on top of each other.

Tile One Window

The Tile One Window command minimizes all but the current window. The current window is grown to fill most of the Source Insight window's workspace area.

Tile Two Windows

The Tile Two Windows command splits the screen into two windows; the first window will contain the current file. The other window will contain the previous current file (i.e. the last file you were viewing). This command is only allowed if two or more windows are open.

Tile Vertical

The Tile Vertical command arranges all windows so they are not overlapping. The tiling algorithm will attempt to make windows taller than they are wide. This is useful for viewing 2 or more files side by side.

Toggle Extend Mode

The Toggle Extend Mode command toggles extend mode on and off. If extend mode is on, then movement commands, such as Cursor Up, Cursor Down, Top of Window, and Go To Line, (to name a few), will cause the current selection to be extended to the new location. If extend mode is off, then the movement commands simply put an insertion point at their destination.

Toggle Insert Mode

The Toggle Insert Mode toggles between insertion and overstrike mode. In insertion mode, characters typed will be inserted before the insertion point. In overstrike mode, characters typed will replace characters at the insertion point.

Top of File

The Top of File command moves the insertion point to the first line in the current file.

Top of Window

The Top of Window command moves the insertion point to the first line in the active window.

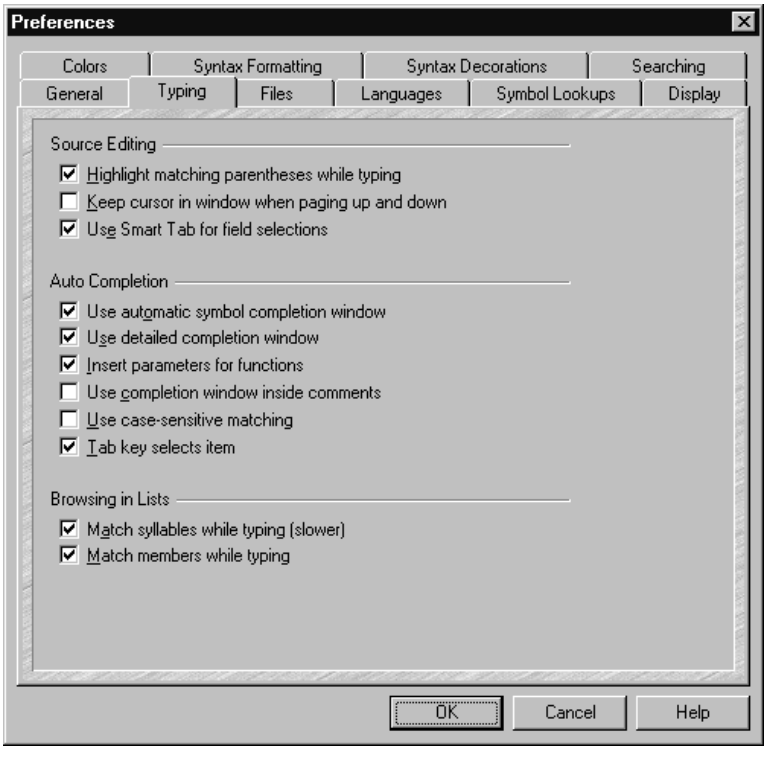
Touch All Files in Relation

This command is available from the right-click menu of the Relation Window. It touches (i.e. updates the last-write timestamp) on all the files currently shown in the Relation Window.

This is useful if you want to cause re-compilation of all the files containing the symbols shown in the Relation Window. This is especially handy if the Relation Window is showing references.

Typing Options

This command specifies typing and editing options. It activates the Typing page of the Preferences dialog box.



Highlight matching parentheses while typing

Source Insight will momentarily highlight up to the matching parentheses or brace when you type a closing parentheses or brace. This is useful for seeing that you are matching up your braces and parentheses while typing new source code.

Always use Symbol Windows

Source Insight will automatically attach a symbol window to each new window it opens, if the file is setup to be parsed for symbols in the Document Options dialog box. Each document type can also control whether it uses a symbol window.

Keep cursor in window when paging up and down

If enabled, then the insertion point cursor will stay visible in the window as you page up and down. If disabled, then the insertion point will remain at its position in the text regardless of how you scroll.

Use Smart Tab for field selections

Causes the regular Tab key to invoke the Smart Tab command. See Smart Tab.

Auto Completion

This section customizes the way the auto completion feature works.

The auto completion window appears after you type a character. Its purpose is to reduce the number of characters you have to type by automatically providing a list of possible identifiers to insert. The symbols that appear in the completion window depend on the context, such as the cursor position, or the type of variable being referred to.

Use automatic symbol completion window

Enables the auto completion window. If this option is disabled, then you can still use auto completion by invoking it manually with the Complete Symbol command.

Use detailed completion window

The auto completion window shows details about functions, such as their parameter lists.

Insert parameters for functions

The parameter list for a function is inserted along with the name of the function. This only happens if there isn't a parenthesized list of parameters already to the right of the insertion point.

Use completion window inside comments

Enables the auto completion feature while you are typing into a comment. If this option is disabled, then you can still use auto completion by invoking it manually with the Complete Symbol command.

Use case-sensitive matching

The auto completion window will list only symbol names that match the case as you have typed it. If this option is disabled, then all symbols that match your input, regardless of case, are listed. However, Source Insight will attempt to select the item that matches the case you typed.

Tab key selects item

If checked, then pressing the Tab key while the completion window is showing will insert the selected item in the list. If this option is disabled, then a Tab key just inserts a tab. In any case, the Enter key can also be used to insert the selected item.

Browsing in Lists

This section customizes the way that auto completion works while typing in symbol and file lists.

Match syllables while typing

Enables syllable matching by default. If this option is disabled, you can still match syllables by inserting a space character first, followed by your input.

Match members while typing

Class and struct members are matched according to your input. A full symbol name is like a path that starts with the symbol's top-most container. For example, a class member might be named `MyClass.member`. Enabling this option will allow you to just type "member". If disabled, then only full symbol names are matched. You can still use this feature, even if disabled, by prefixing your input with a dot ('.'), like this:

```
.member
```

Undo

The Undo command reverses the action of the last editing command in the current file. For example, if you type some new text and perform Undo, the text you typed is removed. Alternatively, if you delete a line and perform Undo, the line reappears.

In effect, as you edit, Source Insight makes a list of the changes you've made to each file. The Undo command backs up through that list and the Redo command advances through the list.

Undo information is saved for each open file independently.

The Undo command does not undo cursor movements, as with some other editors. In Source Insight, the best way to "back up" through the history of cursor movements is to use the Go Back command (and conversely, the Go Forward command).

You can use the Undo command several times in a row to undo several changes. You can also use the Undo All command to undo all changes made to the current file since the last time it was opened.

The undo history is maintained after you save a file, as long as the file is open. Undo history is lost when the file is closed, or if you use the Checkpoint command to perform a final save. You can control whether undo is normally available after saving by using the Preferences: Files dialog box.

It is possible to save a file, and then undo the few last edits. After that, the file saved on disk represents a state that has more edits than the current file buffer. This can become confusing. To help, Source Insight displays the file name with an asterisk whenever the loaded buffer differs from the file saved on disk. If you try to undo to a point before the file was saved, you will prompted to confirm you want to do that.

Undo All

The Undo All command undoes all the editing changes to the current file since the last time it was opened. See Also: Undo, Redo, Redo All commands.

Vertical Scroll Bar

This command toggles the vertical scroll bar in the current on and off.

If you want all windows to have a vertical scroll bar or not, you should use the Preferences: Display Options dialog box option: **Vertical Scroll Bar for each new window**.

View Clip

(On Clip Window tool bar and right-click menu)

The View Clip command displays the selected clip in a source file window. If you close the window, you will be asked if you want to retain the clip in the Context Window.

View Relation Outline

This command changes the Relation Window so it displays its tree data in a textual outline format. See “Relation Window” on page 69.

View Relation Window Horizontal Graph

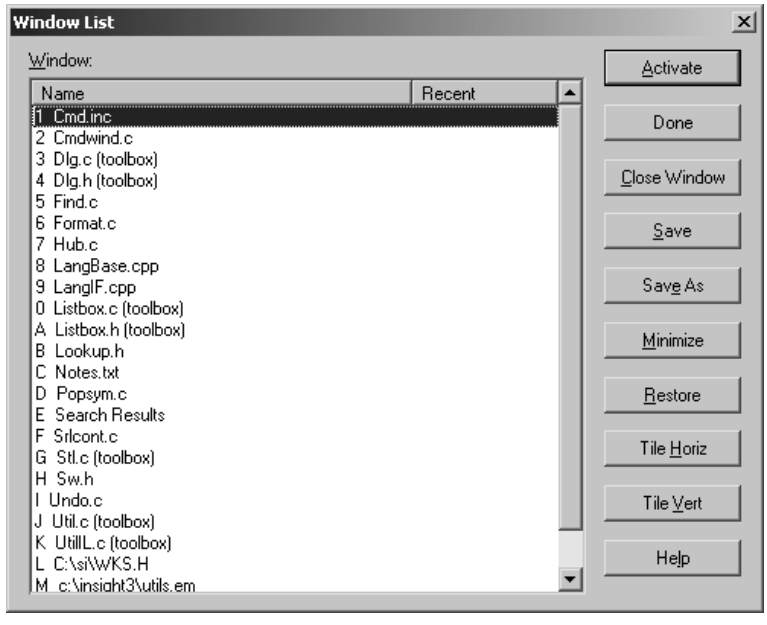
This command changes the Relation Window so it displays a tree graph view that grows horizontally, from left to right. See “Relation Window” on page 69.

View Relation Window Vertical Graph

This command changes the Relation Window so it displays a tree graph view that grows vertically, from top to bottom. See "Relation Window" on page 69.

Window List

Manages the list of open source windows and file buffers.



Window list

Contains a list of all open source windows, and file buffers that may be open in the "background" that do not appear in a window. You can sort the window list by name or by recently-used order. To sort the list, click on the header titles at the top of the list.

Activate

Brings the selected window to the front.

Close Window

Closes the selected windows.

Save

Saves the selected files.

Save As

Peforms a Save As command on each selected file.

Minimize

Minimizes the selected windows.

Restore

Restores the selected windows.

Tile Horiz

Tiles the selected windows horizontally so they are generally wider than they are tall

Tile Vert

Tiles the selected windows vertically so they are generally taller than they are wide.

Word Left

The Word Left command moves the insertion point to left by one word.

Word Right

The Word Right command moves the insertion point to the right by one word.

Zoom Window

If the current window is not already maximized, the Zoom Window command maximizes it. If the current window is already maximized, the Zoom Window command restores the window to its un-maximized size.

Macro Language Guide

This chapter describes the Source Insight macro language. The Source Insight macro language syntax is similar to C. This chapter assumes you are familiar with basic programming concepts.

Overview

Source Insight provides a C-like macro language, which is useful for scripting commands, inserting specially formatted text, and automating editing operations.

Macros are saved in a text file with a .EM extension. The files are added to your project, or to any project on the Project Symbol Path, or to the Base project. Once a macro file is part of the project, the macro functions in the file become available as commands in the Key Assignments or Menu Assignments dialog boxes.

Basic Syntax Rules

Source Insight's macro language is not case sensitive. As with most other languages, white space is ignored. Semi-colons are not required, but are ignored. Variable names must begin with a letter, not a digit.

Macro Functions

A macro function is declared in a format that looks like a C function. White space is ignored. Macros have the form:

```
macro my_macro()
{
        /* comments */
        .. statements..   // comments
}
```

Macro functions can have parameters and can call other macros. You can return a value from a macro by using "return n" where n is the return value. For example:

```
macro add2(n)
{
        return n + 2
}
```

Macro Scopes and References

All macros exist at the same global scope and all macros declared in any open file, stored in a project, or in a project on the project symbol path are in scope. That is, you can have forward references to macros. You do not have to declare them before calling them.

Source Insight uses its symbol lookup engine to resolve references to macro names when macros are executed and when the user invokes a macro command. Therefore, symbol lookup rules apply to macro name binding. See "Symbols and Projects" in the "Projects" chapter for more information on symbol lookups.

You can also use the various symbol lookup techniques to locate macro functions while you edit. (See "Finding Symbols in your Project" in the "Projects" chapter.) For example, you can type the name of a macro in the Browse Project Symbols dialog box and jump to its definition.

Running Macros

You can run macros by invoking the macro command directly with a keystroke or menu item, or by using the Run Macro command to begin running macro statements at the current cursor position.

Macros as Commands

Source Insight considers macro functions that have no parameters to be user level commands. Macro commands appear in the command lists of the Key Assignments and Menu Assignments dialog boxes. This allows you to place macro commands on the menu or in the keymap. You can also run commands directly from those dialog boxes. If a macro function has parameters, it will not appear as a command in the command lists, and you wont be able to assign a keystroke to it or put it on a menu.

If you create a new macro command function in a macro file, you must save the macro file and allow Source Insight to synchronize it with the project database files before the macro command will appear in menu and key assignments command lists.

You can also store macros in the Base project, or any other project on the project symbol path. Source Insight will search those projects when resolving macro names.

Note: Source Insight will not add macro functions to the user level command list unless they are saved in Source Insight macro files using a .EM extension.

Running Inline Macro Statements

The Run Macro command starts executing macro statements starting at the line the cursor is on. This allows you to run open

coded macro statements in any type of file. This is very useful for testing and debugging macro code.

Running inline macro statements is also useful for running short utility macro scripts stored inside of a program comment. Remember to use the Stop statement at the end, or the macro interpreter will attempt to run past the end of the comment.

For example, this inline macro inside a C file comment searches for references to the identifier `ucmMax` and causes recompilation of all the files that refer to it. To use it, the user places the cursor on the first line of the macro and invokes the Run Macro command.

```
#define ucmMax     120

/* Macro: touch all ucmMax references:

// to run, place cursor on next line and invoke "Run Macro"
hbuf = NewBuf("TouchRefs") // create output buffer
if (hbuf == 0)
        stop
SearchForRefs(hbuf, "ucmMax", TRUE)
SetCurrentBuf(hbuf) // put search results buffer in a window
SetBufDirty(hbuf, FALSE); // don't bother asking to save it
Stop
*/
```

Statements

Macros are composed of individual statements. Groups of two or more statements are enclosed with curly braces { }.

A statement can be one of the following:

- A macro language statement element, such as an `if` or `while` statement. These statements are described later.
- A call to a user-defined macro function. You define and save macro functions in a macro source file.
- A call to an internal macro function, such as GetCurrentBuf. Source Insight provides many built-in functions. They are described in a later section.
- An invocation of a Source Insight user command, such as Line_Up or Copy. All user-level commands in Source Insight are available to the macro language. To call a user command, use the name of the command by itself, without parameters. Parentheses are not used. If the command contains embedded spaces in the command name, you must replace the spaces with underscore (_) characters. For example, to call the "Select All" command, the statement should look like `Select_All`.

Function and variable names are not case sensitive.

Statement syntax is generally the same as in C or Java, except that semi-colons are not required, and are ignored. If your fingers are used to typing them, you don't have to change your habits.

Here are a few example statements:

```
hbuf = OpenBuf("file.c")      // call internal function
OpenBuf
SaveBufAs(hbuf, "filenew.c")  // call internal function
SaveBufAs
Select_All     // call user-level command "Select All"
Copy           // call user-level command "Copy"
Line_Up        // call user-level command "Line Up"
x = add2(n)    // call user-defined macro function add2.
```

You can stop execution using the "`stop`" statement.

If you run a Source Insight command with a dialog box from a macro, the dialog box just comes up.

The following table summarizes the basic macro language statements.

Statement	Description
`break`	Exits a while loop.
`if (cond)… else`	Tests a condition.
`continue`	Continues a while loop at the top.
`return n`	Returns n from a function.
`stop`	Stops executing the macro.
`while (cond)`	Loops while `cond` is true.

Variables

You define variables by simply assigning a value to them. They do not need to be declared. However, you can use the `var` statement to explicitly declare a variable.

Variable names are not case sensitive. Variable names must begin with a letter or underscore, not a digit.

All variables are string variables. There are no types like in C, and no need to declare variables. This makes working with variables very easy. There is no need to do conversions or casts. In addition, there is no need for string memory management.

Declaring a Variable

The `var` statement declares a single local variable.

```
var variable_name
```

You don't have to declare variables, but there are a couple of benefits:

- Syntax formatting works for references to it.
- Variables are initialized to the empty string value, so they are not confused with literal values when used without being initialized.

For example:

```
macro SomeFunction()
{
    var localx

    // "localx" is displayed with "Ref to Local" style
    localx = 0
}
```

Variable Name Expansion

Identifiers are expanded to their string value if the identifier is the name of a defined variable; otherwise, they are used literally. They are also used literally if they are surrounded by double quotes. For example:

```
s = abc  // same as s = "abc" if abc is not defined
```

..or..

```
abc = Hello
s = abc  // same as s = "Hello" (if Hello is not defined!)
s = "abc"    // s equals "abc"
```

To avoid unintentionally using the name of variable as its value, you should get into the habit of declaring your variables with the `var` statement.

Expanding Variables in a String

You can insert a variable into another string by using the special `@` character. When a variable name appears inside a literal string, and the variable name is surrounded by `@` characters, then Source Insight replaces the `@variable@` with the variable value.

For example:

```
S = "Hey, @username@, don't break the build again!"
```

This example replaces `@username@` with the value of the variable `username` in the string.

You can escape the `@` character with a backslash `\` or by using two `@` characters together. For example:

```
S = "Mail info@@company.com for information."
```

Variable Arithmetic

Even though variables are represented as strings, you can perform arithmetic on them. E.g.

```
s = "1"
x = s + 2      // x now contains the string "3"
y = 2 * x + 5  // x now contains "11"
```

Using variables numerically is very natural and normally you don't have to even think about how they are stored. The only time you need to be careful is if a variable might contain a string that does not evaluate to a number. In that case, an error is generated. For example:

```
s = "hello"
x = s + 1  // error
```

You can tell if a string is numeric by calling the IsNumber function.

Floating-point numbers are not supported.

Indexing Into Strings

You can index into variables as though they are character arrays. Indices are zero-based. For example:

```
s = "abc"
x = s[0]    // x now contains the string "a"
```

Indexing to a variable yields a new string containing a single character.

The string that is returned by indexing **one past** the last character is an empty string. This is similar to zero-terminated strings in C, only the zero "character" is an empty string.

```
s = "abc"
length = strlen(s)
ch = s[length] // ch now contains the empty string
if (ch == "")
        msg "End of string."
```

Record Variables

While C data structures are not supported, aggregate record variables are. A record variable is actually a delimited list of "name=value" pairs. Record variables are used in the same way that a "struct" would be used in C.

Record variables are returned by some internal macro functions because it is a convenient way to return multiple values.

Record fields are referred to with the dot (.) operator using a `<recordvar>.<fieldname>` format.

For example, this reads the name of a symbol's file location from a symbol lookup record variable.

```
Filename = slr.file     // get file field of slr
LineNumber = slr.lnFirst  // get lnFirst field of slr
```

You assign values to a record variable in a similar way:

```
userinfo.name = myname; // set "name" field of userinfo
```

You can initialize an empty record by assigning `nil` to it:

```
userinfo = nil         // make a new empty record
userinfo.name = "Jeff"     // begin adding fields
```

Record Variable Storage

Record variables are stored simply as strings. Each field is stored as a "fieldname=value" pair, delimited with semi-colons.

For example,

```
name="Joe Smith";age="34";experience="guru"
```

If you wanted to construct a whole record variable string from scratch, you would have to surround it in double quotes and escape each embedded quote mark with the backslash character, like this: (C programmers should be used to this.)

```
rec = "name=\"Joe Smith\";age=\"34\";experience=\"guru\""
```

However, it is just easier to assign a field at a time to it. For example:

```
Rec = nil  // initializes as an empty string
Rec.name = "Joe Smith"
Rec.age = "34"
Rec.experience = "guru"
```

The fields in the record do not have to be in any particular order. There is no pre-declared structure associated with record variables, so you are free to attach new fields whenever you want by simply assigning a value to them.

For example:

```
Location = GetSymbolLocation(symname)
Location.myOwnField = xyz  // append a field when you feel
like it!
```

Array Techniques

The Source Insight macro language does not support array variables. However, file buffers can be used as dynamic arrays. Each line in a buffer can represent an array element. Furthermore, record variables can be stored in each line to give you record arrays.

Buffer functions are described in a following section. Some useful functions are NewBuf and CloseBuf for creating and destroying buffers; and the buffer line functions: GetBufLine, PutBufLine, InsBufLine, AppendBufLine, DelBufLine, and GetBufLineCount. You can also call NewWnd to put the array buffer in a window so you can see the array contents.

This example creates a buffer array of records.

```
hbuf = NewBuf()
rec = "name=\"Joe Smith\";age=\"34\";experience=\"guru\""
AppendBufLine(hbuf, rec)
rec = "name=\"Mary X"\";age=\"31\";experience=\"intern\""
AppendBufLine(hbuf, rec)
// hbuf now has 2 records in it
…
rec = GetBufLine(hbuf, 0) // retrieve 0th element
CloseBuf(hbuf)
```

Special Constants

Some constant values are always defined while a macro is running. As with all other identifiers, the constant names are not case sensitive.

Constant	Value
`True`	"1"
`False`	"0"
`Nil`	"" – the empty string.
`hNil`	"0" – an invalid handle value.
`invalid`	"-1" – an invalid index value.

Operators

Most binary operators are the same as in C. Operator precedence is the same as C. You can use parentheses to group expressions.

Operator	Meaning
`+ and.`	add and subtract
`* and /`	multiply and divide
`!`	Invert or "Not". E.g. !True equals False.
`||`	logical OR operation
`&&`	logical AND operation
`!=`	logical NOT EQUAL operation
`==`	logical EQUAL operation
`<`	less than
`>`	greater than
`<=`	less than or equal to
`>=`	greater than or equal to
`#`	string concatenation
`"@var@"`	variable expansion. used inside of quoted strings to expand a variable in the string.

Conditional operators are treated thus for variables:

Operator	Meaning
`==`	strings must be equal (case sensitive)
`!=`	strings must not be equal (case sensitive)
`<,>,<=,>=`	strings converted to numbers and then compared. Empty strings or strings that are non-numeric result in a zero value.

Conditions and Loops: if-else *and* while

The Source Insight macro language supports `if` and `while` statements.

The if Statement

The if statement is used to test a condition and execute a statement or statements based on the condition. The if statement has the following syntax:

```
if (condition)
        statement       // condition is true
```

In the above example, if condition is true, then statement is executed. You can use {} brackets to group more than one statement. For example:

```
if (condition)
        {
        statement1
        statement2
        }
```

An else clause can be added to the if statement:

```
if (condition)
        statement1      // condition is true
else
        statement2      // condition is false
```

The while statement

The while statement loops while a condition is true. The while statement has the following syntax:

```
while (condition)
        statement  // keeps executing statement until
condition is false
```

In the above example, the statement in the while block is executed as long as the condition is true. The condition is tested at the top of the loop each time. You can use {} brackets to group more than one statement. For example:

```
while (condition)
        {
        statement1
        statement2
        }
```

Break and Continue

The break statement is used to exit out of a while loop. If more then one while loop is nested, the break statement breaks out of the innermost loop.

```
while (condition)
        {
        if (should_exit)
            break   // exit the while loop
        …
        }
```

The continue statement continues again at the top of the loop, just before condition expression is evaluated.

```
while (condition)
        {
        if (skip_rest_loop)
            continue    // continue at the top of the while
loop
        …
        }
```

Conditional Evaluation

Source Insight evaluates the **whole** conditional expression each time. This is an important difference from the way that C conditional expressions are evaluated. In C, the expression may be partially evaluated. Consider the following conditional expression:

```
if (hbuf != hNil && GetBufLineCount(hbuf) > x)
```

This conditional expression would lead to an error in Source Insight if hbuf is equal to hNil. In C, the evaluation would be terminated after determining that hbuf != hNil. However, in Source Insight, the whole expression is evaluated. In this case, causing an error since hNil would be passed as the buffer handle to GetBufLineCount.

In Source Insight, this statement would have to written like this:

```
if (hbuf != hNil)
        if (GetBufLineCount(hbuf) > x)
```

Internal Macro Functions

Source Insight provides many built-in internal macro functions. There are functions for manipulating strings, file buffers, windows, projects, and symbol information.

The syntax for calling an internal macro function is the same as for calling a user-defined macro function. For example:

```
hbuf = GetCurrentBuf()    // call GetCurrentBuf function
```

Naming Conventions

Variables and function parameters described in this section are named using the following conventions.

Name	Meaning
s and sz	a string
ch	single character string. If more than one character is in a string, only the first character is used.
ich	zero-based index to a character in a string or character in a line
ichFirst	first index in a range of characters
ichLast	last index in a range of characters (inclusively)
ichLim	limit index - one past the last index in a range
cch	count of characters
fThing	"f" means flag or boolean. fThing means "Thing" is True.
TRUE	a non-zero value, e.g. "1"
FALSE	a zero value, i.e. "0"
ln	zero-based line number
lnFirst	first line number in a range
lnLast	last line number in a range (inclusively)
lnLim	limit - one past the last line number in a range
hbuf	handle to a file buffer
hwnd	handle to a window
hprj	handle to a project
hsyml	Handle to a symbol list
Any other names	general string variables

String Functions

cat (a, b)

Concatenates strings a and b together and returns the result.

strlen (s)

Returns the length of the string.

isupper (ch)

Returns TRUE if the given character *ch* is uppercase. If *ch* is a string with more than one character, only the first character is tested.

islower (ch)

Returns TRUE if the given character *ch* is lowercase. If *ch* is a string with more than one character, only the first character is tested.

toupper (s)

Returns the uppercase version of the given string.

tolower (s)

Returns the lowercase version of the given string.

strmid (s, ichFirst, ichLim)

Returns the middle string of *s* in the range from *ichFirst* up to, but not including *ichLim*. That is, *s*[*ichFirst*] through s[*ichLim - 1*]. If ichFirst equals ichLim, then an empty string range is specified.

strtrunc (s, cch)

Returns string *s* truncated to *cch* count of characters.

AsciiFromChar (ch)

Returns the ASCII value of the given character *ch*. If *ch* is a string with more than one character, only the first character is tested.

CharFromAscii (ascii_code)

Returns a string containing a single character that corresponds to the ASCII code in *ascii_code.*

IsNumber (s)

Returns TRUE if the string *s* contains a numeric string. Non numeric strings are converted to "0" when used in arithmetic expressions.

User Input and Output Functions

Ask (prompt_string)

Prompts the user with a message box window displaying the string *prompt_string*. The Ask message box has an OK button, and a Cancel button. Clicking the Cancel button stops the macro.

Msg (s)

Display a message window showing the string *s*. The message box has a Cancel button that the user can click to stop the macro.

StartMsg (s)

Display a message window showing the string *s*. The message box has a Cancel button that the user can click to stop the macro. The message window stays up after returning.

EndMsg ()

Takes down the message box started by StartMsg.

Beep ()

Gives a single beep.

GetKey ()

Waits for the user to press a key and returns the key code. The key code is a special numeric value that Source Insight associates with each key. You can use the CharFromKey function to map a key code into a character.

GetChar ()

Waits for the user to press a key and returns a single character.

CharFromKey (key_code)

Returns the character associated with the given key code. Returns zero if the *key_code* is not a regular character key code.

key_code is a numeric keyboard value that is returned by GetKey and KeyFromChar. You can use CharFromKey to convert a key_code into a character.

FuncFromKey (key_code)

Return the function key number (1 - 12 for F1 - F12) from a function key code. Returns zero if *key_code* is not a function key code.

key_code is a numeric keyboard value that is returned by GetKey and KeyFromChar. You can use CharFromKey to convert a key_code into a character.

IsFuncKey (key_code)

Returns TRUE if *key_code* is a function key, or FALSE if not.

key_code is a numeric keyboard value that is returned by GetKey and KeyFromChar. You can use CharFromKey to convert a key_code into a character.

IsCtrlKeyDown (key_code)

Returns TRUE if the CTRL key is down for *key_code*. Note: key codes contain the CTRL and ALT key state.

key_code is a numeric keyboard value that is returned by GetKey and KeyFromChar. You can use CharFromKey to convert a key_code into a character.

IsAltKeyDown (key_code)

Returns TRUE if the ALT key is down for *key_code*. Note: key codes contain the CTRL and ALT key state.

key_code is a numeric keyboard value that is returned by GetKey and KeyFromChar. You can use CharFromKey to convert a key_code into a character.

GetSysTime(fLocalTime)

Returns a SYSTIME record, which contain string representations of the time and date.

If fLocalTime is non-zero, then the local time is returned, otherwise the system time (expressed in Coordinated Universal Time (UTC)) is returned.

The **SYSTIME** record has the following fields:

Field	Description
time	the time of day in string format.
date	the day of week, day, month, and year as a string.
Year	current year.
Month	current month number. January is 1.
DayOfWeek	day of week number. Sunday is 0, Monday is 1, etc.
Day	day of month.
Hour	current hour.
Minute	current minute.
Second	current second
Milliseconds	current milliseconds

CmdFromKey(key_value)

Returns the string name of the Source Insight command currently mapped to key_value. The command returned is the name of the command that gets invoked when the user presses key_value.

key_value is a numeric keyboard value that is returned by GetKey and KeyFromChar.

You can use CharFromKey to convert a key_value into a character.

Example:

```
key = GetKey();
cmd_name = CmdFromKey(key);
msg("That key will invoke the @cmd@ command.");
```

AssignKeyToCmd(key_value, cmd_name)

Assigns the key_value to command named by cmd_name. Subsequently, when the user presses the key_value, the command is invoked.

key_value is a numeric keyboard value that is returned by GetKey and KeyFromChar. You can use CharFromKey to convert a key_value into a character.

cmd_name is the string name of the command.

Example:

```
key = GetKey();
AssignKeyToCmd(key, "Open Project");
```

KeyFromChar(char, fCtrl, fShift, fAlt)

Returns a key value, given a character and modifier key states. A key value is a numeric keyboard value that is returned by GetKey. You can use CharFromKey to convert a key value into a character.

Inputs:

- *char* - the character part of the keystroke. It is not case sensitive.
- *fCtrl* - non-zero if the CTRL key is included.
- *fShift* - non-zero if the Shift key is included.
- *fAlt* - non-zero if the ALT key is included.

The *char* parameter can have some special values:

Char Value	Meaning
a-z	Simple alpha characters
Fx	Function Key number x; e.g. F10
Nx	Numeric keypad character x; e.g. N+ for "+" key
Up, Down, Left, Right	Arrow keys
Page Up, Page Down	Other special keys
Insert, Delete	
Home, End,	
Tab, Enter	

Example:

```
key = KeyFromChar("c", 1, 0, 0);  // Ctrl+C
```

```
AssignKeyToCmd(key, "Page Down");
key = KeyFromChar("F9", 0, 0, 1);  // Alt+F9
AssignKeyToCmd(key, "Close Project");
```

Key Examples:

```
key = GetKey()
if (IsFuncKey(key))
        Msg cat("Function key = ", FuncFromKey(key))
if (IsAltKeyDown(key))
            Msg "Alt key down"
if (IsCtrlKeyDown(key))
        Msg "Ctrl key down"
ch = CharFromKey(key)
if (Ascii(ch) == 13)
        Msg "You pressed Enter!"
else if (toupper(ch) == 'S')
        Search_Files
...
```

Buffer List Functions

A buffer list is a collection of file buffer handles. There is only one buffer list in the Source Insight application. It contains the file buffer handles for all open source files. You can use the buffer list functions to enumerate through all file buffers.

BufListCount ()

This function returns the number of buffers in the buffer list. Use BufListItem to access the buffer handle at a particular index position.

BufListItem (index)

This function returns the buffer handle at the given index. The size of the buffer list is returned by BufListCount. Index values start at zero, and continue up to one less than the value returned by BufListCount.

This example enumerates all the open buffer handles:

```
cbuf = BufListCount()
ibuf = 0
while (ibuf < cbuf)
        {
        hbuf = BufListItem(ibuf)
        // … do something with buffer hbuf
        ibuf = ibuf + 1
        }
```

File Buffer Functions

File buffer functions are used to create and manipulate file buffers and the text within them. A file buffer is the loaded image of a text file. File buffers are edited by the user and then saved back to disk with the Save command.

Many of the file buffer functions use buffer handles (hbuf). These are handles to open file buffers. An hbuf is typically a small integer value. An hbuf value of hNil (zero) indicates an error.

NewBuf (name)

Creates a new empty file buffer and returns a handle to the file buffer (an hbuf). The name of the new buffer is specified by the *name* parameter. NewBuf returns hNil if the buffer could not be created due to errors.

OpenBuf (filename)

Opens a file named *filename* into a file buffer and returns a handle to the file buffer. OpenBuf returns hNil if the file could not be opened.

OpenMiscFile (filename)

Opens a file named *filename*. The action taken depends on the type of files opened. For example, opening a file with a .CF3 extension will load a new configuration file. Opening a project file (.PR extension) will open that project. OpenMiscFile returns TRUE if it was successful, or FALSE if not.

CloseBuf (hbuf)

Closes a file buffer. *Hbuf* is the buffer handle.

SaveBuf (hbuf)

Saves a file buffer to disk. *Hbuf* is the buffer handle.

SaveBufAs (hbuf, filename)

Saves a file buffer a different file name. *Hbuf* is the buffer handle. *Filename* is the name of the new file.

GetCurrentBuf ()

Returns a handle to the current buffer. The current buffer is the file buffer that appears in the front-most source file window. Returns hNil if there is no current buffer (i.e. no open source file windows).

SetCurrentBuf (hbuf)

Sets the active file buffer to the buffer whose handle is *hbuf*. The current buffer is the file buffer that appears in the front-most source file window.

GetBufHandle (filename)

Returns the handle of the open file buffer whose name is *filename*. This function searches all open file buffers to find the one that matches the given filename parameter. GetBufHandle returns hNil if no buffer can be found with the given file name.

ClearBuf (hbuf)

Empties the buffer *hbuf* so that it contains no lines.

GetBufName (hbuf)

Returns the name of the file associated with a file buffer. *Hbuf* is the file buffer handle.

GetBufLineCount (hbuf)

Returns the number of lines of text in a file buffer. *Hbuf* is the file buffer handle.

GetBufLineLength (hbuf, ln)

Returns the number of characters on the line *ln* in the given file buffer *hbuf*.

GetBufLine (hbuf, ln)

Returns the text of the line *ln* in the given file buffer *hbuf*.

PutBufLine (hbuf, ln, s)

Replaces the line of text for line number *ln* in the file buffer *hbuf* with the string in *s*.

InsBufLine (hbuf, ln, s)

Inserts a new line of text *s* for line number *ln* in the file buffer *hbuf*.

AppendBufLine (hbuf, s)

Appends a new line of text *s* to the file buffer *hbuf*.

DelBufLine (hbuf, ln)

Deletes the line *ln* from the file buffer *hbuf*.

CopyBufLine (hbuf, ln)

Copies the line *ln* from the file buffer *hbuf* to the clipboard.

PasteBufLine (hbuf, ln)

Pastes the clipboad contents just **before** the line *ln* in the file buffer *hbuf*.

SetBufIns (hbuf, ln, ich)

Sets the cursor position insertion point to line number *ln* at character index *ich* in file buffer *hbuf*. A macro error is generated if the given file buffer is not already displayed in a source file window.

GetBufLnCur (hbuf)

Returns the current line number of the user's selection inside of the file buffer *hbuf*. A macro error is generated if the given file buffer is not already displayed in a source file window.

GetBufSelText (hbuf)

Returns the selected characters in the file buffer *hbuf* as a string. A maximum of one line of text is returned. This is useful for getting the text of a word selection. A macro error is generated if the given file buffer is not already displayed in a source file window.

SetBufSelText (hbuf, s)

Replaces the currently selected characters in the file buffer *hbuf* with the string *s*. This is useful for replacing the text of a word selection. A macro error is generated if the given file buffer is not already displayed in a source file window.

This is the simplest way to insert new text into a buffer. For example, this code inserts "new text" into the current buffer at the current insertion position:

```
hbuf = GetCurrentBuf()
SetBufSelText(hbuf, "new text")
```

MakeBufClip (hbuf, fClip)

If *fClip* is True, this turns the file buffer *hbuf* into a Clip buffer. Following this, the buffer will appear as a regular clip in the Clip Window.

If *fClip* is False, this turns the buffer into a regular, non-clip file buffer.

Clip buffers are automatically saved to the Clips subdirectory of the Source Insight program directory when Source Insight exits.

GetBufProps (hbuf)

Returns a Bufprop record, which contains properties for the given buffer.

Bufprop Record

A Bufprop record contains file buffer properties. It is returned by GetBufProps.

The Bufprop record has the following fields:

Field	Description
Name	The buffer name (i.e. file name)
fNew	True if buffer is a new, unsaved buffer.
fDirty	True if the buffer has been edited since it was saved or opened.
fReadOnly	True if the buffer is read-only.
fClip	True if the buffer is a clip that appears in the Clip Window.
fMacro	True if the buffer is a macro file.
fRunningMacro	True if the buffer is a currently running macro file.
fCaptureResults	True if the buffer contains captured custom command output.
fSearchResults	True if the buffer contains search results.
fProtected	True if the buffer protected and reserved for internal use.
lnCount	The line count of the buffer.
Language	The programming language of the buffer. The language is determined by the file's document type.
DocumentType	The document type of the buffer.

IsBufDirty (hbuf)

Returns True if the buffer is dirty. A dirty buffer is one that has been edited since it was opened or saved. A dirty buffer contains changes that have not been saved.

IsBufRW (hbuf)

Return True if the given buffer is read-write-able. This function returns False if the buffer is read-only.

SetBufDirty (hbuf, fDirty)

Sets the dirty state of the given buffer to *fDirty*. A dirty buffer is one that has been edited since is was opened or saved. A dirty buffer contains changes that have not been saved.

When the user closes a dirty buffer, Source Insight prompts to save the file. You can use this function to un-dirty a buffer so that the user is not prompted to save it.

RenameBuf (hbuf, szNewName)

Renames the given buffer to *szNewName*. The file on disk is also renamed. If the buffer is a member of an open project, then the file is renamed in the project.

Environment Functions

PutEnv (env_name, value)

Sets the environment variable named *env_name* to *value*.

GetEnv (env_name)

Returns the value of the environment variable given in *env_name*. Returns an empty string if the environment variable does not exist.

SetReg (reg_key_name, value)

Sets the value associated with the registry key named *reg_key_name*. The key value is stored under the key path: HKEY_CURRENT_USER/Software/Source Dynamics/Source Insight/2.0. The key is created if it doesn't exist already.

The SetReg and GetReg functions give you a way to store your own Source Insight related information between sessions.

GetReg (reg_key_name)

Returns the value associated with the registry key named *reg_key_name*. The key value is stored under the key path: HKEY_CURRENT_USER/Software/Source Dynamics/Source Insight/2.0. Returns an empty string if the key does not exist.

IsCmdEnabled (cmd_name)

Returns TRUE if the command specified in *cmd_name* is currently enabled. A command would not be enabled if Source Insight cannot run it due the state of the program. For example, the Save command is not enabled if there are no open source file windows.

RunCmd (cmd_name)

Runs the command specified by *cmd_name*. This allows you to run special commands, namely custom commands, which are defined with the Custom Commands command.

Window List Functions

A window list is a collection of source file window handles. There is only one window list in the Source Insight application. It contains all the source window handles. You can use the window list to enumerate through all source file windows.

WndListCount ()

This function returns the number of windows in the window list. Use WndListItem to access the window handle at a particular index in the list.

WndListItem (index)

This function returns the window handle at the given index. The size of the window list is returned by WndListCount. Index values start at zero, and continue up to one less than the value returned by WndListCount.

This example enumerates all the open window handles:

```
cwnd = WndListCount()
iwnd = 0
while (iwnd < cwnd)
        {
        hwnd = WndListItem(iwnd)
        // … do something with window hwnd
        iwnd = iwnd + 1
        }
```

Window Functions

Window functions use window handles (hwnd). These are macro handles to open source file windows. Windows display file buffers in them. An hwnd is typically a small integer value. An hwnd of hNil indicates an error.

Note that a macro hwnd is similar in concept, but is not exactly the same as a window handle HWND in the Microsoft Windows API.

Each window has a *selection* in it, which describes what characters are selected. See the GetWndSel function for more information.

NewWnd (hbuf)

Creates a new window and displays the file buffer *hbuf* in the window. NewWnd returns a window handle, or hNil if errors.

CloseWnd (hwnd)

Closes the window *hwnd*. Closing a window does **not** close the file buffer displayed in the window.

SetCurrentWnd (hwnd)

Sets the front-most active window. *Hwnd* is the window handle to activate.

GetCurrentWnd ()

Returns the handle of the active, front-most source file window, or returns hNil if no windows are open.

GetWndHandle (hbuf)

Returns a window handle for the front-most window that displays the file buffer specified by *hbuf*. GetWndHandle returns hNil if the file buffer is not in a window.

Since a file buffer may appear in more than one window, GetWndHandle searches through all windows in front-to-back

order. So if the specified file buffer is the current buffer in the active window, the handle for that window is always returned.

GetWndBuf (hwnd)

Returns the handle of the file buffer displayed in the window *hwnd*.

GetWndDim (hwnd)

Returns a DIM record, which describes the pixel dimension of the given window *hwnd*.

The horizontal dimension returned is the width of the text area of the window only. It does not include the left margin or the symbol window attached to the source window.

DIM Record

The DIM record describes horizontal and vertical pixel dimensions.

Field	Description
Cxp	Count of X-pixels (horizontal size)
Cyp	Count of Y-pixels (vertical size)

GetWndLineWidth (hwnd, ln, cch)

Returns the width of a specified line of text in the given window.

Inputs:

Parameter	Description
Hwnd	The window.
Ln	The line number that contains the text to be measured. If ln is out of range, then –1 is returned.
Cch	The count of characters to measure on the line. If cch is set to –1, then the whole line length is measured.

This function allows you to measure the width of characters in a given window. Since the font used by each window is determined by the file's document type, the width of text can vary from window to window. Syntax formatting also affects the width of text.

This function can be used along with ScrollWndHoriz to scroll a window to show a particular character.

Examples

To find the width of the whole line at line 100:

```
Dim = GetWndLineWidth(hwnd, 100, -1)
Msg ("Line 100 is " # dim.cxp # " pixels wide.")
```

To find the width of the first 3 characters on line 200:

```
Dim = GetWndLineWidth(hwnd, 200, 3)
```

GetNextWnd (hwnd)

Returns the next window handle in the window Z order after *hwnd*. This is usually the previous window that was active. GetNextWnd returns hNil if there are no other windows.

For example, if hwnd is the top-most window, then GetNextWnd(hwnd) will return the next window down. Note that if you are using SetCurrentWnd to set the front-most active window, subsequent calls to GetNextWnd are affected.

GetWndSel (hwnd)

Returns the selection state of the window specified by *hwnd*. The selection state is returned in a Selection record.

Selection Record

A Selection record describes the state of a text selection in a window.

The Selection record has the following fields:

Field	Description
lnFirst	the first line number
ichFirst	the index of the first character on the line lnFirst
lnLast	the last line number
ichLim	the limit index (one past the last) of the last character on the line given in lnLast
fExtended	TRUE if the selection is extended to include more than one character. FALSE if the selection is a simple insertion point. Note: this is the same as the following expression: (sel.fRect || sel.lnFirst != sel.lnLast || sel.ichFirst != sel.ichLim)
fRect	TRUE if selection is rectangular (block style). FALSE if the selection is a linear range of characters.
The following fields **only** apply if fRect is TRUE:	
xLeft	the left pixel position of the rectangle in window coordinates.
xRight	the pixel position of the right edge of the rectangle in window coordinates.

SetWndSel (hwnd, selection_record)

Sets the selection state for the window specified by *hwnd* to the Selection record given in *selection_record*. See GetWndSel for information on the Selection record.

GetWndSelLnFirst (hwnd)

Returns the first line number of the selection in the window *hwnd*.

GetWndSelLnLast (hwnd)

Returns the last line number of the selection in the window *hwnd*.

GetWndSelIchFirst (hwnd)

Returns the index of the first character in the selection in the window *hwnd*.

GetWndSelIchLim (hwnd)

Returns the index of one past the last character in the selection in the window *hwnd*.

MaximizeWnd (hwnd)

Maximizes (or "zooms") the window specified by *hwnd*.

MinimizeWnd (hwnd)

Minimizes (or "iconizes") the window specified by *hwnd*.

ToggleWndMax (hwnd)

Toggles the window *hwnd* between maximized and restored sizes.

IsWndMax (hwnd)

Returns TRUE if the window *hwnd* is currently maximized.

IsWndMin (hwnd)

Returns TRUE if the window *hwnd* is currently minimized.

IsWndRestored (hwnd)

Returns TRUE if the window *hwnd* is currently not maximized and not minimized.

ScrollWndToLine (hwnd, ln)

Scrolls the window *hwnd* to show the line number *ln* at the top of the window.

ScrollWndVert (hwnd, line_count)

Scrolls the window *hwnd* vertically by an amount given in *line_count*.

If *line_count* is less than zero, then the scroll is backward in the file (screen contents scrolls down).

If *line_count* is greater than zero, then the scroll is forward in the file (screen contents scrolls up).

ScrollWndHoriz (hwnd, pixel_count)

Scrolls the window *hwnd* horizontally by an amount given in *pixel_count*.

If *pixel_count* is less than zero, then the scroll is backward in the line (screen contents scrolls right).

If *pixel_count* is greater than zero, then the scroll is forward in the line (screen contents scrolls left).

GetWndVertScroll (hwnd)

Returns the vertical scroll state of the window *hwnd*. The vertical scroll state is the line number that appears at the top of the window.

GetWndHorizScroll (hwnd)

Returns the horizontal scroll state of the window *hwnd*. The horizontal scroll state is the pixel count of the scroll.

GetWndLineCount (hwnd)

Returns the vertical size of the window *hwnd* in lines. This is the maximum number of lines potentially visible in the window. If the file buffer does not fill the entire window, GetWndLineCount will still return the maximum number of lines.

Bookmark Functions

All bookmarks are kept in a single bookmark list. You can use the bookmark functions to enumerate through all bookmarks, and to add and remove bookmarks. The bookmark list is persisted in the workspace file.

Bookmark Record

A bookmark is a position in a file buffer. The Bookmark record has the following fields:

Field	Description
Name	The bookmark name.
File	The file name of the bookmark position.
ln	The line number position.
ich	The character index on the line.

BookmarksCount ()

This function returns the number of bookmarks in the bookmark list. Use BookmarksItem to access the bookmark at a particular index in the list.

BookmarksItem (index)

This function returns the bookmark at the given index. The size of the bookmark list is returned by BookmarksCount. Index values start at zero, and continue up to one less than the value returned by BookmarksCount.

This example enumerates all the bookmarks:

```
cmark = BookmarksCount()
imark = 0
while (imark < cmark)
       {
       bookmark = BookmarksItem(imark)
       // … do something with bookmark
       imark = imark + 1
       }
```

BookmarksAdd (name, filename, ln, ich)

Adds a new bookmark. The new bookmark name is in *name*. The bookmark position is in the file *filename* at line number *ln* at character index *ich*.

Returns True if successful, or False if errors.

BookmarksDelete (name)

Deletes the bookname named *name*.

BookmarksLookupName (name)

Searches for the bookmark named *name*.

Returns a Bookmark record or nil if the bookmark is not found.

BookmarksLookupLine (filename, ln)

Searches for the bookmark at the given position. The file is in *filename* and the line number is *ln*.

Returns a Bookmark record, or nil if the bookmark is not found.

Symbol List Functions

A symbol list is a zero-based indexed collection of Symbol records.

Some of the symbol access functions return symbol list handles.

Symbol lists are allocated resources that should be freed using SymListFree when you are finished accessing them. Source Insight automatically cleans up dynamically allocated resources when a macro terminates. However, Source Insight may run out of resources if you allocated many handles without freeing unused handles.

SymListCount ()

This function returns the number of symbols in the symbol list. Use SymListItem to access the symbol record at a particular index in the list.

SymListItem (hsyml, isym)

This function returns the Symbol record at the zero-based index *isym* in the symbol list *hsyml*. The size of the symbol list is returned by SymListCount. See "Symbol Record" on page 415.

Index values start at zero, and continue up to one less than the value returned by SymListCount.

This example enumerates all symbols in the symbol list:

```
csym = SymListCount(hsyml)
isym = 0
while (isym < csym)
        {
        symbol = SymListItem(isym)
        // … do something with symbol
        Msg ("symbol name = " # symbol.name)
        isym = isym + 1
        }
```

SymListInsert (hsyml, isym, symbolNew)

This function inserts a symbol record into the symbol list *hsyml*. The symbol is inserted just before *isymBefore*. If *isymBefore* is –1, then the

symbol is appended to the end of the list. The symbol record is given in *symbolNew*.

SymListRemove (hsyml, isym)

Removes an element from the symbol list *hsyml*. The symbol element at *isym* is deleted.

SymListNew ()

Allocates a new, empty symbol list. Returns the new symbol list handle, or hNil if errors. You should call SymListFree when you are finished with the symbol list.

SymListFree (hsyml)

This function deallocates the given symbol list.

Symbol Functions

Symbol functions allow you to access Source Insight's symbol lookup engine. Source Insight maintains symbolic information about your project in a symbol database. These symbol functions make use of the symbol database and Source Insight's built-in language parsers to locate symbols in your source files.

You may want to review the section “Symbols and Projects” in the “Projects” chapter for a description of how Source Insight maintains symbolic information and what the lookup rules are.

Symbol Record

The Symbol record describes a symbol declaration. It specifies the location and type of a symbol. It is used to uniquely describe a symbol in a project, or in an open file buffer.

Symbol records are returned by several functions, and Symbol records are used as input to several functions.

The Symbol record has the following fields:

Field	Description
Symbol	The full symbol name. A symbol name is actually a path. Every symbol name is divided into path components, which are separated by dot (.) characters. For example, a symbol name might be "myclass.member1". In this example, "member1" is contained by "myclass".
Type	The symbol's type (e.g. "Function", "Class", etc.)
Project	The full path of the project where the symbol was found
File	The full path of the file where the symbol was found
lnFirst	The first line number of the symbol declaration
lnLim	The limit line number of the symbol declaration
lnName	The line number where the symbol's name appears in the declaration.
ichName	The character index of the symbol's name in the declaration at the line lnName.
Instance	The instance number path of the symbol within File. For example, the first occurrence of a symbol is instance 0, the second is instance 1, and so on.

SymbolChildren (symbol)

Returns a new symbol list handle containing the children of the given symbol. The children of a symbol are the symbols declared within the body of the symbol. For example, the children of a class are the class members.

symbol contains a Symbol record.

You should call SymListFree to free the symbol list handle returned by SymbolChildren.

You can use the Symbol List functions to access the symbol list returned by this function.

Example

This example looks up the definition of a symbol and displays its children:

```
symbolname = Ask("What symbol do you want to locate?")
symbol = GetSymbolLocation(symbolname)
if (symbol == nil)
        Msg (symbolname # " was not found")
else
        {
        hsyml = SymbolChildren(symbol)
        cchild = SymListCount(hsyml)
        ichild = 0
        while (ichild < cchild)
            {
            childsym = SymListItem(hsyml, ichild)
            Msg (childsym.symbol # " was found in " #
childsym.file #
                " at line " # childsym.lnFirst)
            ichild = ichild + 1
            }
        SymListFree(hsyml)
        }
```

SymbolParent (symbol)

Returns a Symbol record of the parent of the given symbol. The parent of a symbol is the symbol that contains it.

symbol contains a Symbol record.

SymbolStructureType (symbol)

Returns a Symbol record of the structural type of the given symbol. The structural type is the struct or class type of the symbol, which may be indirectly referenced through typedefs.

symbol contains a Symbol record.

SymbolDeclaredType (symbol)

Returns a Symbol record of the declared type of the given symbol.

symbol contains a Symbol record.

SymbolLeafName (symbol)

Returns the "leaf", or right-most component of the symbol's name.

symbol contains a Symbol record.

Every symbol name is divided into path components, which are separated by dot (.) characters. For example, a symbol name might

be “myclass.member1”. In this example, “member1” is contained by “myclass”.

SymbolContainerName (symbol)

Returns the container component of the symbol's name.

symbol contains a Symbol record.

Every symbol name is divided into path components, which are separated by dot (.) characters. For example, a symbol name might be “myclass.member1”. In this example, “member1” is contained by “myclass”.

SymbolRootContainer (symbol)

Returns the root, or left-most component of the symbol's name.

symbol contains a Symbol record.

Every symbol name is divided into path components, which are separated by dot (.) characters. For example, a symbol name might be “myclass.member1”. In this example, “member1” is contained by “myclass”.

GetSymbolLocation (symbol_name)

Returns the location of the symbol name specified in *symbol_name*. The location is returned in a Symbol record. An empty string is returned if the symbol is not found.

This function performs a look up operation the same way that Source Insight looks up symbols when you use the Jump To Definition command. If the symbol is not found in the current project, or any open file, then all the projects on the project symbol path are searched as well. If more than one declaration is found for *symbol_name*, then the user is presented with a multiple-definition list to select from.

You can also call GetSymbolLocationEx for more control over how the lookup operation is performed, and to locate multiple definitions of the same symbol name.

This example looks up the definition of a symbol and displays its source file and line number.

```
symbol = Ask("What symbol do you want to locate?")
loc = GetSymbolLocation(symbol)
if (loc == "")
        Msg (symbol # " was not found")
else
        Msg (symbol # " was found in " # loc.file #
                " at line " # loc.lnFirst)
```

Locating File Names

GetSymbolLocation can also look up file names. By giving a simple file name as the *symbol_name* parameter, GetSymbolLocation can look up the file in the project, or on the project symbol path, and return the fully qualified path to the file in the location.file field. This can be useful for expanding a simple file name into its full path.

For example:

```
loc = GetSymbolLocation("simple.c")
fullfilename = loc.file
// fullfilename could be something like "d:\proj\simple.c"
```

GetSymbolLocationEx (symbol_name, output_buffer, fMatchCase, fLocateFiles, fLocateSymbols)

Finds all the declarations for the symbol specified in *symbol_name*. Each declaration location is appended as a line to the given buffer *output_buffer* as a Symbol record. If *fMatchCase,* then the symbol name case must match exactly. If *fLocateFiles*, then file names in the project or on the project symbol path are located. File extensions don't have to be specified. If *fLocateSymbols*, then symbol definitions are located. Both *fLocateFiles* and *fLocateSymbols* may be set to True.

Since record variables can be expressed as a string, a Symbol record can be written as a line of text in a buffer. To read a Symbol record from the output buffer, use the GetBufLine function to return the entire line text; in this case a Symbol record. See GetSymbolLocation for a description of the Symbol record.

GetSymbolLocationEx returns the number of matching declarations, or zero if none were found.

Unlike the GetSymbolLocation function, this function will find multiple declarations that match on *symbol_name*. You can use this function to enumerate through each location by scanning each line of the output buffer.

This example looks up a symbol and enumerates through each declaration found:

```
symbol = Ask("What symbol do you want to locate?")
hbuf = NewBuf("output")
count = GetSymbolLocationEx(symbol, hbuf, 1, 1, 1)
ln = 0
while (ln < count)
       {
       loc = GetBufLine(hbuf, ln)
       msg (loc.file # " at line " # loc.lnFirst)
       ln = ln + 1
       }
CloseBuf(hbuf)
```

Locating File Names

GetSymbolLocationEx can also look up file names. By giving a simple file name as the *symbol_name* parameter, GetSymbolLocationEx can look up the file in the project, or on the project symbol path, and return the fully qualified path to the file in the location.file field. See GetSymbolLocation for an example.

If *fLocateFiles* is TRUE, and a symbol name is given without an extension, GetSymbolLocationEx will locate files that have this base name, regardless of extension. For example, if you specify "dlg" in *symbol_name*, GetSymbolLocationEx may return matches on "dlg.c" (a file), and "dlg.h" (another file). Furthermore, if *fLocateSymbols* is also TRUE, then "DLG" (a data type) may also be returned.

GetSymbolLocationFromLn(hbuf, ln)

Returns the Symbol record of the symbol that exists at line number ln within the buffer hbuf. The symbol at ln is a symbol whose declaration includes the given line number ln.

JumpToLocation (location_record)

Jumps to the location given in *location_record*. This opens the file in the Symbol record and moves the cursor to the symbol defined there. This works the same way as the Jump To Definition command.

The Symbol record is returned by the GetSymbolLocation function.

GetCurSymbol ()

Returns the name of the symbol where the current selection is. The current selection is the selection (or cursor position) in the active window. GetCurSymbol returns an empty string if no symbol is found.

GetSymbolLine (symbol_name)

Returns the line number of the symbol named *symbol_name*. If multiple symbols are defined with the same name, then the user will be able to select the appropriate one.

JumpToSymbolDef (symbol_name)

Jumps to the definition of the symbol named *symbol_name*. This opens a file and moves the cursor to the symbol defined there. This works the same way as the Jump To Definition command.

GetBufSymCount(hbuf)

Returns the number of symbols declared in the buffer *hbuf*. Returns zero if no symbols are declared, or if the file could not be processed, or if the document type for the file does not specify a language parser.

GetBufSymLocation(hbuf, isym)

Returns the Symbol record of the symbol indexed by *isym* in the buffer *hbuf*. Each parsed file buffer maintains an index of symbols defined in it. The index is sorted by symbol name. Symbol index values start at zero and go up to the count returned by GetBufSymCount minus one. This function maps a symbol index (*isym*) into a symbol Symbol record.

See GetSymbolLocation for a description of the Symbol record.

GetBufSymName(hbuf, isym)

Returns the name of the symbol indexed by *isym* in the buffer *hbuf*. Each parsed file buffer maintains an index of symbols defined in it. The index is sorted by symbol name. Symbol index values start at zero and go up to the count returned by GetBufSymCount minus one. This function maps a symbol index (*isym*) into a symbol name.

This example iterates through all file buffer symbols:

```
isymMax = GetBufSymCount (hbuf)
isym = 0
while (isym < isymMax)
        {
        symname = GetBufSymName (hbuf, isym)
        ..
        isym = isym + 1
        }
```

Searching Functions

These functions search for references to words and patterns.

SearchForRefs (hbuf, word, fTouchFiles)

Searches for references to the word string in *word* throughout the whole project. Each line that contains *word* is appended to the buffer *hbuf*. If *fTouchFiles* is TRUE, then each file that contains *word* will have its last-modified time stamp set to the current time.

This function is similar to the "Lookup References" command. *Word* can contain more than one word, but this function is much faster if it is a single word.

This example creates a new search results file and searches for references.

```
macro LookupRefs (symbol)
{
        hbuf = NewBuf("Results") // create output buffer
        if (hbuf == 0)
            stop
        SearchForRefs(hbuf, symbol, 0)
        SetCurrentBuf(hbuf) // put buffer in a window
}
```

GetSourceLink (hbufSource, lnSource)

Returns the destination of a source link in a Link record. The source buffer is *hbufSource* and the source line number is *lnSource*. If the given line does not contain a source link, then an empty string is returned.

The destination link points to a location in some file at some line number. This source link information links two arbitrary locations. For example, the Search Results buffer contains source links for each line that matches the search pattern.

Link Record

A Link record describes a location in a file, which is pointed to by a source link.

The Link record has the following fields:

Field	Description
File	The file name
ln	The line number - this is only valid for as long as the file is unchanged. If lines are inserted or deleted from the file, the line number is going to be off. However, you can call GetSourceLink again to get an updated line number.

SetSourceLink (hbufSource, lnSource, target_file, lnTarget)

Creates a new source link. The link source buffer is *hbufSource*. The link source line number is *lnSource*. The link target file is given as a path string in *target_file*. The link target line number is *lnTarget*.

Returns True if successful, or False if not. Target_file does not have to exist. The operation will not fail just because target_file does not exist. Also, target_file does not need to be open.

For consistent results, target_file should contain a fully qualified path name for a file. However, you may pass a simple file spec to this function and it will expand target_file based on what files are included in the current project and on the project symbol path.

Source Links are destroyed when the source buffer closes, or when the source line is deleted.

SearchInBuf (hbuf, pattern, lnStart, ichStart, fMatchCase, fRegExp, fWholeWordsOnly)

Searches for *pattern* in the buffer *hbuf*. The search starts at line number *lnStart* and character index *ichFirst*. SearchInBuf returns a Sel record which spans the matching text. If nothing is found, then an empty string is returned. See GetWndSel for a description of the Sel record.

If *fMatchCase*, then the search is case sensitive.

If *fRegExpr*, then the pattern contains a regular expression. Otherwise, the pattern is a simple string.

If *fWholeWordsOnly* then only whole words will cause a match.

LoadSearchPattern (pattern, fMatchCase, fRegExp, fWholeWordsOnly)

Loads the search pattern used for the Search, Search Forward, and Search Backward commands.

The search pattern string is given in *pattern*.

If *fMatchCase*, then the search is case sensitive.

If *fRegExpr*, then the pattern contains a regular expression. Otherwise, the pattern is a simple string.

If *fWholeWordsOnly* then only whole words will cause a match.

ReplaceInBuf (hbuf, oldPattern, newPattern, lnStart, lnLim, fMatchCase, fRegExp, fWholeWordsOnly, fConfirm)

Performs a search and replace operation in the given buffer.

The search pattern string is given in *oldPattern*.

The replacement pattern string is given in *newPattern.*

The line range is specified by *lnSart* to *lnLim*. The replacements only take place on lines *lnStart* up to *lnLim* - 1.

If *fMatchCase*, then the search is case sensitive.

If *fRegExpr*, then the pattern contains a regular expression. Otherwise, the pattern is a simple string.

If *fWholeWordsOnly* then only whole words will cause a match.

If *fConfirm* then the user will be prompted before each replacement.

Project Functions

Project functions allow you to open and close projects, and get project information.

AddFileToProj(hprj, filename)

Adds the given *filename* to the project *hprj*.

RemoveFileFromProj(hprj, filename)

Removes the given *filename* from the project *hprj*. The file on disk is not altered or deleted.

NewProj (proj_name)

Creates a new project and returns a project handle (hprj), or returns hNil if errors.

OpenProj (proj_name)

Opens the project named *proj_name* and returns a project handle (hprj), or hNil if errors.

CloseProj (hprj)

Closes the project *hprj*.

DeleteProj (proj_name)

Delete the project named in *proj_name*. If that project is currently open, then the user is asked if they want to close it first. If the user does not close the project, then the project is not deleted.

GetProjName (hprj)

Returns the name of the project *hprj*. The name contains the full path of the project file.

GetProjDir (hprj)

Returns the root directory path of the project *hprj*.

GetCurrentProj ()

Returns the name of the currently open project. Source Insight only allows the user to open a single project at a time; however from the macro language, more than one project can be open.

GetProjFileCount (hprj)

Returns the number of files added to the project *hprj*.

GetProjFileName (hprj, ifile)

Returns the name of the project file associated with index *ifile* in the project *hprj*.

Each project has an index of project files, sorted by file name. GetProjFileName maps an index to a file name. File index values start at zero and go up to the count returned by GetProjFileCount.

This example interates through all project files:

```
ifileMax = GetProjFileCount (hprj)
ifile = 0
while (ifile < ifileMax)
        {
        filename = GetProjFileName (hprj, ifile)
        ..
        ifile = ifile + 1
        }
```

GetProjSymCount (hprj)

Returns the number of symbols in the project *hprj*.

GetProjSymName (hprj, isym)

Returns the name of the symbol associated with index *isym* in the project *hprj*.

Each project has an index of symbols, sorted by symbol name. GetProjSymName maps an index to a symbol name. Symbol index

values start at zero and go up to the count returned by GetProjSymCount minus one.

This example interates through all project symbols:

```
isymMax = GetProjSymCount (hprj)
isym = 0
while (isym < isymMax)
        {
        symname = GetProjSymName (hprj, isym)
        ..
        isym = isym + 1
        }
```

GetProjSymLocation (hprj, isym)

Returns symbol location information in a Symbol record for the symbol associated with index *isym* in the project *hprj*.

Each project has an index of symbols, sorted by symbol name. GetProjSymLocation maps an index to a symbol Symbol record. Symbol index values start at zero and go up to the count returned by GetProjSymCount.

You can call JumpToLocation to move to the Symbol record returned by GetProjSymLocation.

See GetSymbolLocation for more information on Symbol records.

SyncProj (hprj)

Synchronizes the project *hprj*. All files in the project are checked for external changes and Source Insight's symbol database is updated incrementally for files that have changed.

SyncProjEx (hprj, fAddNewFiles, fForceAll, fSupressWarnings)

Synchronizes the project *hprj*. All files in the project are checked for external changes and Source Insight's symbol database is updated incrementally for files that have changed.

New files are automatically added to the project if *fAddNewFiles* is True. Only file names that match document types defined in the Document Options command are added.

If *fForceAll*, then each file in the project is re-synchronized, regardless of its time stamp.

If *fSuppressWarnings*, then Source Insight will not issue warnings if it has errors opening files.

Miscellaneous Macro Functions

DumpMacroState (hbufOutput)

This function appends the current state of the running macro to the buffer hbufOutput. The macro state consists of the values of all variables, and the execution stack. This function is useful when debugging macros.

GetProgramInfo ()

Returns a ProgInfo structure, which contains information about Source Insight.

ProgInfo Record

The ProgInfo record describes the version of Source Insight that is running. It has the following fields:

Field	Description
ProgramName	The name of the program (i.e. "Source Insight")
versionMajor	The major version number. If the version number is 1.02.0003, then the major version is 1.
versionMinor	The minor version number. If the version number is 1.02.0003, then the major version is 2.
versionBuild	The build number of the version. If the version number is 1.02.0003, then the build number is 3.
CopyrightMsg	The Source Dynamics copyright message.
fTrialVersion	True if the program is a Trial version.
fBetaVersion	True if the program is a Beta version.
ExeFileName	The name of the executable file.
cchLineMax	The maximum number of characters allowed in a source file line.
cchPathMax	The maximum number of characters supported in a path name.
cchSymbolMax	The maximum number of characters allowed in a symbol's full name.
cchCmdMax	The maximum number of characters allowed in a command, custom command, or macro command name.
cchBookmarkMax	The maximum number of characters allowed in a bookmark name.
cchInputMax	The maximum number of characters allowed in a dialog box text input field.
cchMacroStringMax	The maximum number of characters allowed in a macro string variable.
lnMax	The maximum number of lines supported in a source file.
integerMax	The maximum integer value supported.
integerMin	The minimum integer value supported (a negative number).

GetProgramEnvironmentInfo ()

Returns a ProgEnvInfo record, which contains information about the environment where Source Insight is running.

ProgEnvInfo Record

The ProgEnvInfo record contains information about the environment where Source Insight is running.

Field	Description
ProgramDir	The Source Insight program directory, where the program file is stored.
TempDir	The temporary files directory.
BackupDir	The backup directory, where Source Insight stores backups of files that you save.
ClipDir	The directory where clips are persisted.
ProjectDirectoryFile	The name of the project directory file. The project directory file contains a list of all the projects that have been opened.
ConfigurationFile	The name of the currently active configuration file.
ShellCommand	The name of the command shell. The command shell depends on the operating system version you are running.
UserName	The registered user's name.
UserOrganization	The registered user's organization.
SerialNumber	The registered license serial number.

Other Information about Macros

Debugging

Source Insight does not contain a debugger for macros. However, since macros are interpreted, you can easily figure out what's going on by using the "Msg" function at strategic points in your code to output strings and variable values.

To begin executing a macro statement at the current cursor position, use the Run Macro. Just put the insertion point on the line you want to start running at and invoke Run Macro.

You can dump the execution stack and variable state of a running macro by calling the DumpMacroState function.

Persistence

Variables are not preserved between runs or sessions. However, you can preserve values by storing them in a file, or writing and reading registry keys.

No Self-Modifying Macros

Make sure that a macro is not modifying itself while running. Source Insight will abort any macro that attempts to edit a file containing a running macro.

Sample Macros

A macro file is in included with Source Insight called "utils.em". This file contains some useful functions and you may want to look at it to see some examples.

Appendix: Upgrading From Version 2

This appendix is intended for those who are upgrading from version 2.x of Source Insight.

Upgrading to Version 3 from Version 2

Installing Version 3

If you currently have version 2.x installed on your machine, you should install version 3.1 into a *different* directory than version 2.x.

Opening Older Projects

Source Insight 3.1 can read the old project files created by version 2.x, however it will need to convert them to version 3.1 format. If you plan to keep Source Insight version 2.x around, it may be better to recreate your projects for version 3.1.

Finding Your Old Projects

If you install version 3.1 into a different program directory than version 2.x, version 3.1 will not have a record of your old projects. You can still open them.

To open a version 2.x project, use the Project > Open Project command. In the dialog box, click the Browse button, and navigate your way to the old project's .PR file. Select the old .PR file, click OK, and then click Open. Source Insight will begin converting the project to version 3.1 format.

Loading Old Customizations

Configuration files are used to store your customizations. The file format of configuration files has changed for version 3.1, but Source Insight can still read the 2.x format files. In addition, the file extension has been changed from .CF to .CF3.

To load your old configuration file, use the Options > Load Configuration command. Click the Load button and navigate to the directory where Source Insight version 2.x is installed. You should see files with .CF extensions. The main global configuration file is named global.cf in version 2.x. Select that file and click OK. The old configuration file will be loaded and saved automatically to *global.cf3* – the new name of the global configuration file.

Using Version 3 and Version 2 Together

You can use both version 3.1 and version 2.x together on the same machine. They each use separate registry settings and should not conflict. However, you should follow these guidelines:

- If you currently have version 2.x installed on your machine, you should install version 3.1 into a *different* directory than version 2.x
- Don't run instances of version 2.x and version 3.1 at the same time.
- Creating separate projects for version 3.1 is recommended, although the project files are somewhat upward and downward compatible.
- You can open a version 2.x project, but you will have to click the Browse button in the Open Project dialog box to locate the old .PR file yourself. If you installed version 3.1 in a new directory (recommended), then version 3.1 will have no foreknowledge of the old projects already created.
- You can open your old configuration file with the Options > Load Configuration command. Point to your old 2.x directory and your old *.CF file. Note that new configuration files have a .CF3 file extension.

New Commands

New Command List

The following table summarizes new commands in version 3.1 of Source Insight.

Command	Summary
Activate Relation Window	Opens and selects the Relation Window.
Add and Remove Project Files	Adds and removes files from the current project. This replaces the old "Add Files" and "Remove Files" commands.
Advanced Options	Allows you to enable and disable various internal caches. This is provided for troubleshooting.
Build Project	Custom tool command: Builds the project.
Check In	Custom source control: Checks in the current file.
Check Out	Custom source control: Checks out the current file.
Checkpoint	Saves the current file to disk and erases its change history.
Checkpoint All	Saves all open files to disk and erases their change history.
Clean Build	Custom tool command: Builds the whole project from scratch.
Clear Highlights	Removes all word highlighting in all source windows. Highlighting is applied by using the Highlight Word command.
Color Options	Specifies colors of user interface items.
Compile File	Custom tool command: Compiles the current file.
Drag Line Down	Moves selected text down by one line.
Drag Line Down More	Moves selected text down by several lines.
Drag Line Up	Moves selected text up by one line.
Drag Line Up More	Moves selected text up by several lines.
Edit Condition	Edits the value of the selected parsing condition.
Expand Special	Used inside tree lists: expands the selected item a specified number of tree levels.
General Options	Specifies general preferences.
Go To Next Change, and Go To Previous Change	Moves the cursor to the next or previous block of lines were edited. I.e. it moves to the next or last set of change marks.

Highlight Word	Toggles highlighting of the word under the cursor in all source windows. This is like using a highlighter pen on paper.
HTML Help	Looks up the currently selected word in the HTML Help file.
Incremental Search	Searches incrementally while you type a pattern string.
Incremental Search Backward	Searches backward incrementally while you type a pattern string.
Insert ASCII	Inserts a character by ASCII value.
Jump To Base Type	Jumps to the base structure type of the selected symbol.
Jump To Caller	Jumps to the function that calls the selected function.
Jump To Prototype	Jumps to the function prototype of the selected function.
Keyword List	Edits the keyword list used for syntax formatting the current language.
Language Properties	Edits custom language properties.
Line Numbers	Toggles the display of line numbers.
Lock Relation Window	Toggles Relation Window locking. Its contents do not change when locked.
Lowercase	Converts the selected text to lowercase.
New Relation Window	Creates a new Relation Window. You can have as many Relation Windows as you like. Each window has its own set of options.
Next Relation Window View	Cycles through the view modes of the Relation Window.
Preferences	Lets you specify user options. This one property sheet dialog box contains several tabs for various options, such as Display, Files, and Syntax Formatting.
Project Document Types	Displays project files by document type in the Project Window.
Project File Browser	Displays the File Browser in the Project Window.
Project File List	Displays all project files in the Project Window.
Project Symbol Classes	Displays project symbols by class in the Project Window.
Project Symbol List	Displays all project symbols in the Project Window.
Project Window Properties	Displays the properties of the Project Window.
Recent Files	A submenu contains recently opened file names.
Refresh Relation Window	Updates the Relation Window with the relation for the currently selected symbol.
Relation Graph Properties	Displays the Graphing properties of the Relation Window.
Relation Window	Toggles the Relation Window on and off.

Relation Window Properties	Displays the properties of the Relation Window.
Reload File	Reloads the current file from disk, losing ALL changes since saving.
Run Project	Custom tool command: Run the project executable.
Save A Copy	Saves the current file to a new file, but does not replace or affect the current file.
Search Project	Searches for text or keywords across all project files.
Searching Options…	Specifies options for handling the Search Results.
Setup Common Projects…	Creates common external projects.
Setup HTML Help	Finds the HTML Help file on your disk.
Show Relations	Updates the Relation Window to show information about the selected symbol.
Simple Tab	Inserts a regular tab, overriding the Smart Tab mode.
Smart Beginning of Line	Special version of Beginning of Line command.
Smart End of Line	Special version of End of Line command.
Smart Tab command	When used at various positions, moves the selection to the next "field". A field is defined as "an interesting position in the current context."
Source Dynamics on the Web	Opens the Source Dynamics web site in your Internet browser.
Special Edit	A submenu contains special editing commands.
Style Properties	Sets formatting properties for display styles.
Symbol Lookup Options	Sets options for looking up symbol definitions.
Symbol Window Properties	Displays the properties of the symbol window on the left of each source window.
Sync File to Source Control Project	Custom source control: Gets the latest version of the selected file.
Sync to Source Control Project	Custom source control: Gets the latest version of all project files.
Syntax Formatting	Specifies syntax formatting options for displaying source files.
Toggle Case	Toggles the case of the selected text.
Touch All Files in Relation	Touches all files referenced in the Relation Window.
Typing Options	Specifies typing and editing options.
Undo Check Out	Custom source control: Reverses the check-out of the current file.
Uppercase	Converts the selected text to uppercase.
Window List	Manages the list of source windows.

File Format Compatibility with Older Versions

Version 3.1 is able to open version 2.x project files and some other files, however not all information is copied, and some indexing has to be recreated.

Version 2.x File Type	How Version 3.1 Handles It
Project Files – *.PR	Version 3.1 opens the project and converts it into version 3.1 format. This requires re-indexing the symbol database and may take a little while. When it is finished, you can start looking up symbols. However, it still requires re-parsing the whole project later, either in the background, or using the Project > Synchronize Files command. The nice thing about this conversion is that you don't have to re-add your source files from scratch. Version 2.1 can open a version 3.1 format project, but it will down-convert it again to 2.1 format, which again requires re-indexing the symbol database. If you plan to switch between v 2.x and v 3.1 a lot, it will probably be easier to maintain two separate projects – one in each format. Note that v 3.1 projects have a few more files associated with them, and some extensions have changed. Each project is now made up of the following file extensions: .PR, .PS, .PO, .PFI, .PRI, .IMB, .IMD, .IAB, .IAD
Configuration Files - *.CF	Version 3.1 can open version 2.x configuration files, however the file format is not downward compatible. The default extension has changed from .CF to .CF3 to make it easier to use v 2.x and v 3.1 together. After you install v 3.1, you can use the Options > Load Configuration command to find your old .CF file (normally it's called Global.CF and it is in the version 2.x program directory). Display options stored in a v 2.x configuration file are not converted, but other customizations, like key assignments and menu assignments, are converted. **Note** that v 3.1 now saves the configuration changes automatically, whereas v 2.x used to prompt you when you exited.
Workspace Files - *.WK	Version 3.1 cannot read version 2.x workspace files. Previous 2.x sessions will not be restored when you open them in v 3.1.
Recovery Files - *.RCV	Version 3.1 cannot read version 2.x recovery files. If you had a version 2.x session that crashed and you want to recover it, you must use version 2.x to perform the recovery first.
Project List File – Projects.DB	The project list file name has changed to Projects.DB3 to avoid conflicts with version 2.x.
sihook program component	The sihook.exe program has changed and has been renamed to sihook3.exe to avoid conflicts.

Index

—A—

adding files to a project 47, 174–75, 82, 174–75, 86, 174–75, 177, 340, 437
aliases
  filealias.txt 101, 167
annotations 41, 117, 358
ASP
  uses HTML language support 31–32, 39, 93, 97, 104, 243, 258, 340, 438–39
assignments
  commands to keys 150, 152–54, 172–73, 204, 248–49, 279, 372–73, 442
  commands to menus 150–52, 154, 172, 204, 208, 250, 277, 279, 372–73, 442
auto indent 220, 225, 226, 246
  document types 35–36, 59, 100–101, 88, 100–101, 93, 100–101, 152, 154, 164, 174, 176, 178, 219–20, 269–70, 273, 290, 318, 330, 360, 428, 438
auto-annotations 41, 117, 358
auto-completion 32, 67, 165, 194, 366
  Smart Tab 260, 341, 343, 365, 439
automatic symbol completion 366

—B—

background synchronization 89, 236, 301
background textures 215
backup files 232
boilerplate text 34, 72
  Clip Window 28, 34–35, 51, 58, 65, 67, 72, 167, 190–91, 196, 198, 210, 212, 272, 282, 288, 368, 399–400
bookmark 129, 180–81, 214, 411–12, 431
browsing
  Relation Window 28–29, 40, 50, 66, 67, 69–70, 108, 164, 174, 196, 213, 272, 279–81, 294, 302, 304–7, 364, 368, 69–70, 69–70, 69–70, 437–39
  Symbol Window 27, 35, 50, 56–59, 74, 93, 103, 105, 120, 174, 198, 210, 223–25, 227, 236, 288, 301, 344–45, 354–55, 365, 406, 439
  type resolution 27

—C—

C/C++
  namespace 39, 257
C++ standard library headers
  filealias.txt 101, 167
change marks 35, 194, 214, 240, 321, 437
Clip Window 28, 34–35, 51, 58, 65, 67, 72, 167, 190–91, 196, 198, 210, 212, 272, 282, 288, 368, 399–400
clips 28, 34, 72–73, 167, 191, 210, 399, 432
  Clip Window 28, 34–35, 51, 58, 65, 67, 72, 167, 190–91, 196, 198, 210, 212, 272, 282, 288, 368, 399–400
color formatting 360
colors
  formatting 360
  Syntax Formatting 26, 29, 30–31, 35, 41, 92, 93, 111, 117–18, 359–60, 154, 162, 197, 213, 217, 223, 225, 226, 251–53, 260, 262, 285, 289, 294, 297, 347, 359–60, 377, 406, 438–39
command line options 147
commands
  assignment to keys 150, 152–54, 172–73, 204, 248–49, 279, 372–73, 442
  assignment to menus 150–52, 154, 172, 204, 208, 250, 277, 279, 372–73, 442
common projects 46, 338–40, 338–40, 353, 439
compiler 27, 34, 98, 133, 150–51, 200, 207–8, 238–39, 287
completion

while typing 32, 67, 165, 194
conditional compilation
inactive code 94–95, 98
conditional parsing
global conditions 228, 256
conditions
global 228, 256
project-specific 256
configuration 37, 45, 52, 78, 79, 83, 95, 111–13, 142, 145, 150–56, 168–69, 172, 192, 200, 211, 234, 247–50, 267–69, 271, 277–79, 281–82, 294–95, 323–24, 347–48, 396, 432, 435–36, 442
confirmation dialog 237
Context Window 28, 50, 54, 65–69, 108–9, 164, 173, 195–96, 212, 236, 272, 292, 299, 368
type resolution 27
corrupt projects
Rebuild Project 97, 159, 298, 299
crash recovery 36, 167, 236
rebuild project 97, 159, 298, 299
recovery file 36, 143–44, 230, 236
cross-references 33
current project 34, 46, 61, 79, 82–83, 89–91, 122, 145–47, 174, 177–79, 182, 192, 205, 207, 211, 245–46, 256, 270–72, 279–80, 282, 291, 294–95, 298–99, 310, 313, 318, 322, 329–30, 338, 353, 355, 362, 418, 424, 437
custom commands 34, 54, 86, 150–51, 154, 167, 200–201, 206–7, 236, 249, 269, 277, 287–88, 403
custom language 35, 39, 92–93, 116, 167, 222, 255, 438
customization
key assignments 150, 152–54, 172–73, 204, 248–49, 279, 372–73, 442
menu assignments 150–52, 154, 172, 204, 208, 250, 277, 279, 372–73, 442
customization files
configuration 37, 45, 52, 78, 79, 83, 95, 111–13, 142, 145, 150–56, 168–69, 172, 192, 200, 211, 234, 247–50, 267–69, 271, 277–79, 281–82, 294–95, 323–24, 347–48, 396, 432, 435–36, 442

—D—

database
symbols 26, 40, 62, 78, 79
declarations 79, 88, 98, 109, 163, 361
decorations
display substitutions 357
default text 112, 194
display
background textures 215
change marks 194, 240, 437
default text 112, 194
scroll bars 213
smooth scrolling 213
Syntax Formatting 26, 29, 30–31, 35, 41, 92, 93, 111, 117–18, 359–60, 154, 162, 197, 213, 217, 223, 225, 226, 251–53, 260, 262, 285, 289, 294, 297, 347, 359–60, 377, 406, 438–39
display substitutions 357
document options
margin width 224, 302
tab width 100, 216–18, 224
word wrap 100, 224, 225
document type 59, 61, 63, 82, 88, 92–93, 100–101, 167, 219–25, 243, 252, 255, 290, 302, 354, 365, 400, 406, 421, 438
expand tabs 224, 243
filealias.txt 101, 167
document types 35–36, 59, 100–101, 88, 100–101, 93, 100–101, 152, 154, 164, 174, 176, 178, 219–20, 269–70, 273, 290, 318, 330, 360, 428, 438
Draft View 119, 226
drag and drop 27, 35, 72, 156–57, 271, 281

—E—

editing
change marks 194, 240, 437
options in document types 35–36, 59, 100–101, 88, 100–101, 93, 100–101, 152, 154, 164, 174, 176, 178, 219–20, 269–70, 273, 290, 318, 330, 360, 428, 438
read-only files 233, 320, 343
Smart Rename 31, 41, 90, 110, 122–23, 166, 341

Undo 35, 41, 87, 190, 220, 233, 301, 309, 320–21, 367, 439
word wrap 100, 224, 225
expand tabs 224, 243
extend mode 140, 363
external tools 34

—F—

file buffer 34, 75, 123, 141, 233, 279, 320–21, 368, 387, 395–400, 405, 410–11, 415, 421
file type
document type 59, 61, 63, 82, 88, 92–93, 100–101, 167, 219–25, 243, 252, 255, 290, 302, 354, 365, 400, 406, 421, 438
filealias.txt 101, 167
files
adding to project 47, 174–75, 82, 174–75, 86, 174–75, 177, 340, 437
configuration file 37, 45, 52, 78, 79, 83, 95, 111–13, 142, 145, 150–56, 168–69, 172, 192, 200, 211, 234, 247–50, 267–69, 271, 277–79, 281–82, 294–95, 323–24, 347–48, 396, 432, 435–36, 442
document type of 59, 61, 63, 82, 88, 92–93, 100–101, 167, 219–25, 243, 252, 255, 290, 302, 354, 365, 400, 406, 421, 438
document types 35–36, 59, 100–101, 88, 100–101, 93, 100–101, 152, 154, 164, 174, 176, 178, 219–20, 269–70, 273, 290, 318, 330, 360, 428, 438
file buffer 34, 75, 123, 141, 233, 279, 320–21, 368, 387, 395–400, 405, 410–11, 415, 421
read-only 233, 320, 343
recovery file 36, 143–44, 230, 236
symbol database 26, 40, 62, 78, 79
symbol indexes 33, 104, 159, 272
synchronizing 89, 105, 141–42, 148, 160, 164, 236, 298, 356
synchronizing in the background 89, 236, 301
workspace files 35, 37, 83, 142, 145, 146, 157–58, 157, 168, 230, 271, 281, 324, 411, 442
flat style toolbars 215
floating window 50, 51
fonts
Syntax Formatting 26, 29, 30–31, 35, 41, 92, 93, 111, 117–18, 359–60, 154, 162, 197, 213, 217, 223, 225, 226, 251–53, 260, 262, 285, 289, 294, 297, 347, 359–60, 377, 406, 438–39
function local scope 31, 122, 342
function parameters
inserting 366

—G—

global conditions 228, 256

—H—

HTML 31–32, 39, 93, 97, 104, 243, 258, 340, 438–39
hypertext 33, 74

—I—

inactive code 94–95, 98
Indent Left 243
Indent Right 243
indentation 100, 217, 220, 222, 225, 246
Indent Left command 243
Indent Right command 243
indenting
Smart Tab 260, 341, 343, 365, 439
index
symbols 33, 104, 159, 272
inheritance 27, 32, 39, 70, 306
insert parameters 366
inserting
auto-completion 366
clips 28, 34, 72–73, 167, 191, 210, 399, 432
function parameters 366
Smart Tab 260, 341, 343, 365, 439
symbolic auto-completion 32, 67, 165, 194

—J—

Java 26, 39, 46, 92, 97, 116–17, 180, 222, 257, 338, 340, 375
Jump To Definition
symbol path 46, 89, 90, 163, 164, 338, 353, 362, 372–73, 418–20, 424

—K—

key assignments 150, 152–54, 172–73, 204, 248–49, 279, 372–73, 442
keyboard 37, 58, 79, 135, 150–52, 172, 199, 208, 323, 391–93
Keyword Search 32, 274
Keyword Searches 32, 274
keywords
  Syntax Formatting 26, 29, 30–31, 35, 41, 92, 93, 111, 117–18, 359–60, 154, 162, 197, 213, 217, 223, 225, 226, 251–53, 260, 262, 285, 289, 294, 297, 347, 359–60, 377, 406, 438–39

—L—

language 31, 35–36, 39, 56–57, 82, 88, 92–93, 96–97, 99, 100, 113–14, 116, 121, 152, 154, 162, 167, 219, 222, 251–56, 258–61, 264, 297, 343, 347–48, 359–61, 371–72, 375, 380, 384, 400, 415, 421, 427, 438
  custom 35, 39, 92–93, 116, 167, 222, 255, 438
language support
  Java 26, 39, 46, 92, 97, 116–17, 180, 222, 257, 338, 340, 375
  parser 39, 82, 88, 93–94, 96–97, 98, 111, 114, 121, 221–23, 359, 415, 421
  Syntax Formatting 26, 29, 30–31, 35, 41, 92, 93, 111, 117–18, 359–60, 154, 162, 197, 213, 217, 223, 225, 226, 251–53, 260, 262, 285, 289, 294, 297, 347, 359–60, 377, 406, 438–39
languages
  document types 35–36, 59, 100–101, 88, 100–101, 93, 100–101, 152, 154, 164, 174, 176, 178, 219–20, 269–70, 273, 290, 318, 330, 360, 428, 438
Line Revision Marks 35, 214, 321
Link record 423–24
local scope 31, 122, 342
Lookup References 40, 74, 109, 122–23, 132, 165, 272, 274, 333, 423

—M—

macros
  Link record 423–24
  Symbol record 413–21, 428
margin width 224, 302
matching parentheses 120, 358, 365
memory
  virtual 160–62
menu assignments 150–52, 154, 172, 204, 208, 250, 277, 279, 372–73, 442
menus 37, 79, 107, 150, 156, 172–73, 277–78, 333

—N—

namespace 39, 257
navigating
  source links 33, 74, 122–23, 132–33, 151, 157, 203, 208
Navigating
  with bookmarks 129, 180–81, 214, 411–12, 431
nested parentheses 30, 117
network 33, 45, 80, 85–86, 296
  remote file 85

—O—

operator substitutions 117, 357

—P—

Page Setup 154, 213, 284, 285, 294
paragraph 35, 121, 302, 336
parentheses
  matching 120, 358, 365
  scaled nested parentheses 30, 117
parser 39, 82, 88, 93–94, 96–97, 98, 111, 114, 121, 221–23, 359, 415, 421
parsing
  document types 35–36, 35–36, 59, 100–101, 100–101, 88, 100–101, 100–101, 93, 100–101, 100–101, 152, 154, 164, 174, 176, 178, 219–20, 219–20, 269–70, 269–70, 273, 290, 318, 330, 360, 428, 438
  synchronizing files 89, 105, 141–42, 148, 160, 164, 236, 298, 356
performance tuning 159
printed default text 194
printing
  default text 194

page setup 154, 213, 284, 285, 294
program options
command line 147
project
synchronizing files 89, 105, 141–42, 148, 160, 164, 236, 298, 356
project directory 47, 78, 294–96, 432
Project reports 33
project root directory 80–81, 86, 201, 205, 237, 280, 296, 356
project specific conditions 256
project symbol database 26, 40, 62, 78, 79
project symbol path 46, 89–90, 163, 164, 338, 353, 362, 372, 418–20, 424
common projects 46, 338–40, 338–40, 353, 439
projects
adding files 47, 174–75, 82, 174–75, 86, 174–75, 177, 340, 437
conditional compilation 256
project directory 47, 78, 294–96, 432
project symbol path 46, 89, 90, 163, 164, 338, 347–48, 362, 372–73, 418–20, 424
rebuilding 97, 159, 298, 299
removing files 47, 174–75, 82, 174–75, 86, 174–75, 177, 340, 437
report 33
root directory 80–81, 86, 201, 205, 237, 280, 296, 356
symbol database 26, 40, 62, 78, 79
symbol indexes 33, 104, 159, 272
symbol path 46, 89–90, 163, 164, 338, 353, 362, 372, 418–20, 424
synchronizing in the background 89, 236, 301
workspace files 35, 37, 83, 142, 145, 146, 157–58, 157, 168, 230, 271, 281, 324, 411, 442

—Q—

qualify references 361–62

—R—

read-only files 233, 320, 343
Rebuild Project 97, 159, 298, 299
recovery
from crashes 36, 167, 236
rebuilding projects 97, 159, 298, 299
recovery file 36, 143–44, 230, 236
recovery file 36, 143–44, 230, 236
recovery time 143
Redo 35, 301, 367–68
references
qualifying 361–62
references to symbols 79, 90, 122, 168, 273–74, 343
regular expressions 125, 128, 162, 184, 203, 264–65, 315, 319, 327, 330, 333
Relation Window 28–29, 40, 50, 66, 67, 69–70, 108, 164, 174, 196, 213, 272, 279–81, 294, 302, 304–7, 364, 368, 69–70, 69–70, 69–70, 437–39
remote file 85
renaming symbols
Smart Rename 31, 41, 90, 110, 122–23, 166, 341
reversing changes
Undo 35, 41, 87, 190, 220, 233, 301, 309, 320–21, 367, 439
revision marks 35, 41, 214, 233, 321

—S—

scaled nested parentheses 30, 117
script language 93, 258
scroll bars 213
scrolling
scroll bars 213
smoothly 213
search and replace 34, 74, 122, 157, 272, 425
Search Query
Keyword Searches 32, 274
Search Results 34, 74, 109, 122–23, 132–33, 166, 174, 186, 238–39, 272, 275–76, 326–28, 331, 334, 342, 374, 400, 423, 439
searching
search and replace 34, 74, 122, 157, 272, 425
Search Results 34, 74, 109, 122–23, 132–33, 166, 174, 186, 238–39, 272, 275–76, 326–28, 331, 334, 342, 374, 400, 423, 439
source links 33, 74, 122–23, 132–33, 151, 157, 203, 208
selecting
extending with extend mode 140, 363

selection history 130–31, 138, 157, 237, 337–38
source links 33, 74, 122–23, 132–33, 151, 157, 203, 208
selection history 130–31, 138, 157, 237, 337–38
smart indent 226
auto indent 220, 225, 246
in document types 35–36, 59, 100–101, 88, 100–101, 93, 100–101, 152, 154, 164, 174, 176, 178, 219–20, 269–70, 273, 290, 318, 330, 360, 428, 438
Smart Rename 31, 41, 90, 110, 122–23, 166, 341
Smart Tab 260, 341, 343, 365, 439
smooth scrolling 213
source control 33, 34, 54, 79, 82, 86, 89, 142, 200, 233, 237, 437, 439
commands 86
source links 33, 74, 122–23, 132–33, 151, 157, 203, 208
Link record 423–24
source tips 236
styles 29–31, 41
default text 112, 194
switches
command line options 147
symbol database 26, 40, 62, 78, 79
symbol declarations 79, 88, 98, 109, 163, 361
symbol indexes 33, 104, 159, 272
symbol information
cross-reference 33
project symbol path 46, 89, 90, 163, 164, 338, 353, 362, 372–73, 418–20, 424
Relation Window 28–29, 40, 50, 66, 67, 69–70, 108, 164, 174, 196, 213, 272, 279–81, 294, 302, 304–7, 364, 368, 69–70, 69–70, 69–70, 437–39
synchronizing files 89, 105, 141–42, 148, 160, 164, 236, 298, 356
type resolution 27
Symbol Lookups
Context Window 28, 50, 54, 65–69, 108–9, 164, 173, 195–96, 212, 236, 272, 292, 299, 368
project symbol path 46, 89–90, 163, 164, 338, 353, 362, 372, 418–20, 424
symbol path 46, 89, 90, 163, 164, 338, 347–48, 362, 372–73, 418–20, 424
Symbol record 413–21, 428
symbol references 79, 90, 122, 168, 273–74, 343
qualifying 361–62
Symbol Window 27, 35, 50, 56–59, 74, 93, 103, 105, 120, 174, 198, 210, 223–25, 227, 236, 288, 301, 344–45, 354–55, 365, 406, 439
synchronizing 89, 105, 141–42, 148, 160, 164, 236, 298, 356
Syntax Decorations
annotations 41, 117, 358
display substitutions 357
Syntax Formatting 26, 29, 30–31, 35, 41, 92, 93, 111, 117–18, 359–60, 154, 162, 197, 213, 217, 223, 225, 226, 251–53, 260, 262, 285, 289, 294, 297, 347, 359–60, 377, 406, 438–39
color formatting 360
default text 112, 194

—T—

tab width 100, 216–18, 224
in document types 35–36, 59, 100–101, 88, 100–101, 93, 100–101, 152, 154, 164, 174, 176, 178, 219–20, 269–70, 273, 290, 318, 330, 360, 428, 438
tabs
Smart Tab 260, 341, 343, 365, 439
tab width 100, 216–18, 224
textures
background display 215
time stamps
synchronizing files 89, 105, 141–42, 148, 160, 164, 236, 298, 356
tips
source tips 236
toolbars
flat style 215
tuning
performance 159
type resolution 27
types
of documents 35–36, 59, 100–101, 88, 100–101, 93, 100–101, 152, 154, 164, 174, 176,

178, 219–20, 269–70, 273, 290, 318, 330, 360, 428, 438
typing
  auto-completion 32, 67, 165, 194, 366

—U—

Undo 35, 41, 87, 190, 220, 233, 301, 309, 320–21, 367, 439

—V—

virtual memory 160–62

—W—

windows
  Clip Window 28, 34–35, 51, 58, 65, 67, 72, 167, 190–91, 196, 198, 210, 212, 272, 282, 288, 368, 399–400
  Context Window 28, 50, 54, 65–69, 108–9, 164, 173, 195–96, 212, 236, 272, 292, 299, 368
  floating types 50, 51
  Symbol Window 27, 35, 50, 56–59, 74, 93, 103, 105, 120, 174, 198, 210, 223–25, 227, 236, 288, 301, 344–45, 354–55, 365, 406, 439
word wrap 100, 224, 225
workspace file 37, 83, 142, 145, 146, 157, 168, 230, 271, 281, 324, 411, 442
workspaces 35, 37, 146, 157–58, 324